PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L — A2

(11) International Publication Number: **WO 00/21232**

(43) International Publication Date: 13 April 2000 (13.04.00)

(21) International Application Number: PCT/US99/23008

(22) International Filing Date: 1 October 1999 (01.10.99)

(30) Priority Data:

60/102,957	2 October 1998 (02.10.98)	US
60/117,595	27 January 1999 (27.01.99)	US

(81) Designated States: CA, CN, IL, IN, JP, KR, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(71) Applicant *(for all designated States except US)*: INTERNATIONAL BUSINESS MACHINES CORPORATION [US/US]; Old Orchard Road, Armonk, NY 10504 (US).

(72) Inventors; and

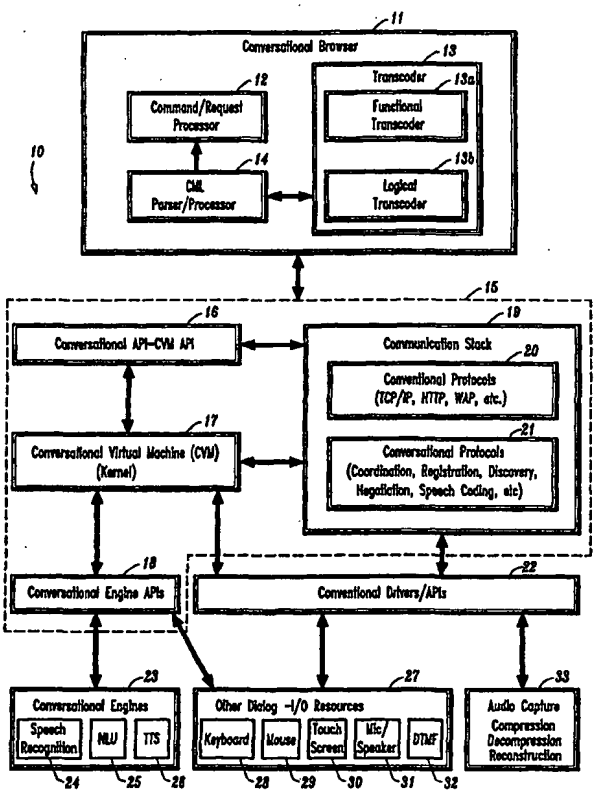
(75) Inventors/Applicants *(for US only)*: GOPALAKRISHNAN, Ponani [IN/US]; 3073 Radcliff Drive, Yorktown Heights, NY 10598 (US). LUCAS, Bruce, D. [US/US]; 2408 Mill Pond Road, Yorktown Heights, NY 10598 (US). MAES, Stephane, H. [BE/US]; 1 Wintergreen Hill Road, Danbury, CT 06811 (US). NAHAMOO, David [IR/US]; 12 Elmwood Road, White Plains, NY 10605 (US). SEDIVY, Jan [CZ/CZ]; U lesa 11, Praha (CZ).

(74) Agent: OTTERSTEDT, Paul, J.; International Business Machines Corporation, Yorktown IP Law Department, T.J. Watson Research Center, Route 134 and Kitchawan Road, Yorktown Heights, NY 10598 (US).

(54) Title: CONVERSATIONAL BROWSER AND CONVERSATIONAL SYSTEMS

(57) Abstract

A conversational browsing system (10) comprising a conversational browser (11) having a command and control interface (12) for converting speech commands or multi-modal input from I/O resources (27) into navigation request, a processor (14) for parsing and interpreting a CML (conversational markup language) file, the CML file comprising meta-information representing a conversational user interface for presentation to a user. The system (10) comprises conversational engines (23) for decoding input commands for interpretation by the command and control interface and decoding meta-information provided by the CML processor for generating synthesized audio output. The browser (11) accesses the engine (23) via system calls through a system platform (15). The system includes a communication stack (19) for transmitting the navigation request to a content server and receiving a CML file from the content server based on the navigation request. A conversational transcoder (13) transforms presentation material from one modality to a conversational modality. The transcoder (13) includes a functional transcoder (13a) to transform a page of GUI to a page of CUI (conversational user interface) and a logical transcoder (13b) to transform business logic of an application, transaction or site into an acceptable dialog. Conversational transcoding can convert HTML files into CML files that are interpreted by the conversational browser (11).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

CONVERSATIONAL BROWSER AND CONVERSATIONAL SYSTEMS

This application is based on provisional applications U.S. Serial Number 60/102,957, filed on October 2, 1998, and U.S. Serial No. 60/117,595 filed on January 27, 1999.

BACKGROUND

1. Technical Field:

The present invention relates generally to systems and methods for accessing information and, more particularly, to a conversational browser that provides unification of the access to various information sources to a standard network protocol (such as HTTP) thereby allowing a pure GUI (graphical user interface) modality and pure speech interface modality to be used individually (or in combination) to access the same bank of transaction and information services without the need for modifying the current networking infrastructure.

2. Description of Related Art:

Currently, there is widespread use of IVR (Interactive Voice Response) services for telephony access to information and transactions. Am IVR system uses spoken directed dialog and generally operates as follows. A user will dial into an IVR system and then listen to an audio prompts that provide choices for accessing certain menus and particular information. Each choice is either assigned to one number on the phone keypad or associated with a word to be uttered by the user (in voice enabled IVRs) and the user will make a desired selection by pushing the appropriate button or uttering the proper word. Conventional IVR applications are typically written in specialized script languages that are offered by manufacturers in various incarnations and for different HW (hardware) platforms. The development and maintenance of such IVR applications requires qualified staff. Conventional IVR applications use specialized (and expensive) telephony HW, and each IVR applications uses different SW (software) layers for accessing legacy database servers. These layers must be specifically designed for each application.

Furthermore, IVR systems are not designed to handle GUI or other modalities other than DTMF and speech. Although it is possible to mix binary data

and voice on a conventional analog connection, it is not possible to do so with a conventional IVR as the receiver. Therefore, IVR systems typically do not allow data/binary input and voice to be merged. Currently, such service would require a separate system configured for handling binary connections (e.g. a form of modem). In the near future, Voice over IP (VoIP) and wireless communication (e.g., GSM) will allow simultaneous transmission of voice and data. Currently, more than one simultaneous call is needed for simultaneous exchange of binary and voice (as it is explained to be useful later to adequately handle specialized tasks) or it will require a later call or callback for asynchronous transmission of the data. This is typically not convenient. In particular, the data exchange can be more than sending or receiving compressed speech and information related to building a speech UI, it can also be the necessary information to add modalities to the UI (e.g. GUI). Assuming that services will be using multiple lines to offer, for example, a voice in / web out (or voice in / web and voice out) modality where the result of the queries and the presentation material also result into GUI material (e.g. HTML displayed on a GUI browser like Netscape Navigator), the service provider must now add all the IT infrastructure and backend to appropriately networked and synchronize its backends, IVR and web servers. A simple but very difficult task is the coordination between the behavior/evolution of the speech presentation material with respect to the GUI or HTML portion of the presentation.

With the rapidly increasing evolution of mobile and home computing, as well as the prevalence of the Internet, the use of networked PCs, NCs, information kiosks and other consumer devices (as opposed to IVR telephony services) to access information services and transactions has also become widespread. Indeed, the explosion of Internet and Intranet has afforded access to virtually every possible information source, database or transaction accessible through what is generally known as a GUI "Web browser," with the conversion of the data and the transactions being performed via proxies, servers and/or transcoders.

In general, a Web browser is an application program (or client program) that allows a user to view and interact with information on the WWW (World Wide Web or the "Web") (i.e., a client program that utilizes HTTP (Hypertext Transfer Protocol) to make requests of HTTP servers on the Internet). The HTTP servers on the Internet include "Web pages" that are written in standard HTML (Hypertext Markup language). An Internet Web page may be accessed from an HTTP server over a packet-switched network, interpreted by the Web browser, and then presented to the user in graphical form. The textual information presented to the user includes

highlighted hyperlinks to new sources of information. The user can then select a hyperlink by, e.g., clicking on the with mouse, to download a new Web page for presentation by the Web browser. The access to legacy databases over the Internet is enabled by several known standards such as **LiveWire** and JDBC (Java Database Connectivity). Furthermore, Web pages can include executable code such as applets (e.g., java programs) that can be downloaded from a server and executed on the browser or on a JVM (java virtual machine) of the system on top of which the browser is built. Other information can be provided by servlets (e.g., java programs) running on the server and pushing changes in the connected browser. The applets and servlets can include CGI (common gateway interface) functions which allow a Web server and applications to communicate with each other. In addition, other information accessing methods include scripts which are predetermined program languages that are interpreted and executed on the browser. This includes, for example, javascripts and DHTML (Dynamic HTML) languages. Plug-ins are programs outside the browser that can be downloaded by the browser and automatically recognized by the browser to run native on the local device and be executed on arguments that are subsequently provided (via download) by the browser. CGI scripts are server side scripts that implement the business logic and produce as output of them running the next presentation material. Applets and plugins can communicate via RMI (remote method invocation), socket connections, RPC (remote procedure call), etc. In addition, complex transcoding schemes, XML (Extensible Markup Language) extensions and scripting languages are used for specific information or services or to simplify the interaction.

As explained above, the purpose of the Internet Web browser and IVR is to access information. The following example describes a typical scenario in connection with a banking application to demonstrate that the paradigm used for accessing the information via IVR with a telephone and via the Internet using a PC and Web browser is similar. For instance, the typical banking ATM transaction allows a customer to perform money transfers between savings, checking and credit card accounts, check account balances using IVR over the telephone. These transactions can also be performed using a PC with Internet access and a Web browser. In general, using the PC, the customer can obtain information in a form of a text menus. In the case of the telephone, the information is presented via audio menus. The mouse clicks on the PC application are transformed to pushing telephone buttons or spoken commands. More specifically, a typical home banking IVR application begins with a welcome message. Similarly,

the Internet home page of the Bank may display a picture and welcome text and allow the user to choose from a list of services, for example:

a. instant account information;

b. transfer and money payment;

c. fund information;

d. check information;

e. stock quotes; and

f. help.

With the IVR application, the above menu can be played to the user over the telephone, whereby the menu messages are followed by the number or button the user should press to select the desired option:

a. "for instant account information, press one;"

b. "for transfer and money payment, press two;"

c. "for fund information, press three;"

d. "for check information, press four;"

e. "for stock quotes, press five;"

f. "for help, press seven;"

The IVR system may implement speech recognition in lieu of, or in addition to, DTMF keys. Let's assume that user wants to get the credit card related information. To obtain this information via the Internet based application, the user would click on a particular hypertext link in a menu to display the next page. In the telephone application, the user would press the appropriate telephone key to transmit a corresponding DTMF signal. Then, the next menu that is played back may be:

a. "for available credit, press one";

b. "for outstanding balance, press two";

c. "if your account is linked to the checking account, you can pay your credit card balance, press three."

Again, the user can make a desired selection by pressing the appropriate key.

To continue, the user may be prompted to provide identification information. For this purpose, the Internet application may display, for example, a menu with an empty field for the user's account number and another for the users social security number. After the information is

filled in it is posted to the server, processed, the replay is formatted and sent back to the user. Over the telephone the scenario is the same. The IVR system may playback (over the telephone) an audio prompt requesting the user to enter his/her account number (via DTMF or speech), and the information is received from the user by processing the DTMF signaling or recognizing the speech. The user may then be prompted to input his/her SSN and the reply is processed in a similar way. When the processing is complete, the information is sent to a server, wherein the account information is accessed, formatted to audio replay, and then played back to the user over the telephone.

As demonstrated above, IVRs use the same paradigm for information access as Web browsers and fulfill the same functionality. Indeed, beyond their interface and modality differences, IVR systems and Web browsers are currently designed and developed as fundamentally different systems. In the near future, however, banks and large corporations will be moving their publicly accessible information sources to the Internet while keeping the old IVRs. Unfortunately, this would require these institutions to maintain separate systems for the same type of information and transaction services. It would be beneficial for banks and corporations to be able to provide information and services via IVR over the Internet using the existing infrastructure. In view of this, a universal system and method that would allow a user to access information and perform transactions over the Internet using IVR and conventional browsers is desired.

SUMMARY OF THE INVENTION

The present invention is directed to a system and method for unifying the access to applications to a standard protocol, irrespective of the mode of access. In particular, the present invention provides a universal method and system for accessing information and performing transactions utilizing, for example, a standard networking protocol based on TCP/IP (such as HTTP (Hypterext Transfer protocol) or WAP (wireless application protocol) and architecture to access information from, e.g., a HTTP server over the Internet such that a pure GUI (graphical user interface) modality and pure speech interface modality can be used individually (or in combination) to access the same bank of transaction and information services without requiring modification of the current infrastructure of currently available networks.

In one embodiment of the present invention, a conversational browser is provided that translates commands over the telephone to an HTTP protocol. The introduction of the conversational browser allows us to unify Internet and Telephone (IVR) and thereby decrease the cost, enlarge the coverage and flexibility of such applications. In particular, for IVR applications, the conversational browser or (telephony browser) can interpret DTMF signaling and/or spoken commands from a user, generate HTTP requests to access information from the appropriate HTTP server, and then interpret HTML-based information and present it to the user via audio messages. The conversational browser can also decode compressed audio which is received from the HTTP server in the HTTP protocol, and play it reconstructed to the user. Conversely, it can capture the audio and transmit it (compressed or not) to the server for distributed recognition and processing. When the audio is captured locally and shipped to the server, this can be done with a plug-in (native implementation) or for example with a java applet or java program using audio and multimedia API to capture the user's input.

For the new proposed IVR architecture and conversational browser, the content pages are on the same HTTP server that are accessed by conventional modes such as GUI browsers, and use the same information access methods, sharing the legacy database access SW layer, etc. In other words, an IVR is now a special case of a HTTP server with a conversational browser. Similar to the conventional GUI browser and PC, the conversational browser, the information and queries will be sent over the switched packed network using the same protocol (HTTP).

The present invention will allow an application designer to set up the application using one framework, irrespective of the mode of access, whether it is through telephone or a WWW browser. All interactions between the application and the client are standardized to the HTTP protocol, with information presented through html and its extensions, as appropriate. The application on the WWW server has access to the type of client that is accessing the application (telephone, PC browser or other networked consumer device) and the information that is presented to the client can be structured appropriately. The application still needs to only support one standard protocol for client access. In addition, the application and content is presented in a uniformed framework which is easy to design, maintain and modify.

In another aspect of the present invention, a conversational browser interprets conversational mark-up language (CML) which follows the XML specifications. CML allows new experienced application developers to rapidly develop conversational dialogs. In another

aspect, CML may follow other declarative syntax or method. Pursuing further the analogy with HTML and the World Wide Web, CML and conversational browser provide a simple and systematic way to build a conversational user interface around legacy enterprise applications and legacy databases.

CML files/documents can be accessed from HTTP server using standard networking protocols. The CML pages describe the conversational UI to be presented to the user via the conversational browser. Preferably, CML pages are defined by tags which are based on the XML application. The primary elements are <page>, <body><menu>, and <form>. Pages group other CML elements, and serve as the top-level element for a CML document (as required by XML). Bodies specify output to be spoken by the browser. Menus present the user with a list of choices, and associate with each choice a URL identifying a CML element to visit if the user selects that choice. Forms allow the user to provide one or more pieces of information, where the content of each piece of information is described by a grammar. The form element also specifies a URL to visit when the user has completed the form.

In another aspect, conversational mark-up language rules can be added by a content provider to an HTML file (or used in place of HTML) to take full advantage of the conversational browser.

In yet another aspect, a conversational transcoder transforms presentation material form one modality to a conversational modality (typically, speech only and/or speech and GUI). This involves functional transformation to transform one page of GUI to a page of CUI (conversational user interface), as well as logical transcoding to transform business logic of an application, transaction or site into an acceptable dialog. Conversational transcoding can convert HTML files into CML files that are interpreted by the conversational browser. The transcoder may be a proprietary application of the server, browser or content provider.

In another aspect, HTML/GUI based structure skeletons can be used to capture the dialog logic or business logic of a GUI site. This information can be used to map the sit, logic or application. After appropriate organization of the dialog flow, each element can undergo functional transcoding into a speech only content or a multi-modal (synchronized GUI and speech interface) page.

In another aspect, a conversational proxy is provided to modify and/or prepare the content description of the application, logic or site to the capabilities of , e.g., the device,

browser and/or engines, preferences of the user or application, load on the servers, traffic on the network, location of the conversational arguments (data files). For instance, the conversational proxy can directly convert proprietary formats such as screen maps of corporate software.

These and other aspects, features and advantages of the present invention will be described and become apparent from the following detailed description of preferred embodiments, which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram of a conversational browsing system according to a preferred embodiment of the present invention;

Fig. 2 is a block diagram of a system for accessing information implementing a conversational browsing system according to an embodiment of the present invention;

Fig. 3 is a block diagram of a system for accessing information implementing a conversational browsing system according to another embodiment of the present invention;

Fig. 4a is a block diagram illustrating a distributed system for accessing information implementing a conversational browsing system according to an embodiment of the present invention;

Fig. 4b is a block diagram illustrating a distributed system for accessing information implementing a conversational browsing system according to another embodiment of the present invention;

Fig. 5 is a block diagram of a conversational information accessing system using conversational markup language according to an embodiment of the present invention;

Fig. 6 is a general diagram of a distributed conversational system using conversational markup language accordance to an embodiment of the present invention;

Fig. 7 is a diagram of an exemplary distributed conversational system using conversational markup language according to an embodiment of the present invention;

Fig. 8 is a diagram of another exemplary distributed conversational system using conversational markup language according to another embodiment of the present invention;

Fig. 9 is a diagram of a yet another distributed conversational information accessing system using conversational markup language according to an embodiment of the present invention; and

Fig. 10 is a diagram of another exemplary distributed conversational information accessing system using conversational markup language according to an embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention is directed to a conversational browsing system and CML (conversational markup language) for building a conversational browsing system using a set of interlinked CML pages. This conversational system is conceptually analogous to building conventional GUI browser applications using a set of interlinked pages written using HTML (hypertext markup language). Moreover, just as HTML provides a set of mechanisms for translating GUI actions into application actions such as visiting other pages or communicating with a server, the conversational browser and CML are used for translating spoken inputs into similar application actions. A CML page describes the conversational UI to be interpreted and presented to the user via the conversational browser. Preferably, CML pages are defined by tags which are based on the current XML (extensible markup language) application (as described in detail below).

It is to be understood that the conversational systems and methods described herein may be implemented in various forms of hardware, software, firmware, special purpose processors, or a combination thereof. In particular, the conversational browser is preferably implemented as an application comprising program instructions that are tangibly embodied on a program storage device (e.g., magnetic floppy disk, RAM, ROM, CD ROM and/or Flash memory) and executable by any device or machine comprising suitable architecture such as personal computers and pervasive computing devices such as PDAs and smart phones.

It is to be further understood that, because some of the constituent components of the conversational browser and other system components depicted in the accompanying Figures are preferably implemented in software, the actual connections between such components may differ depending upon the manner in which the present invention is programmed. Given the teachings herein, one of ordinary skill in the related art will be able to contemplate these and similar implementations or configurations of the present invention.

Conversational Browser Architecture

Referring now to Fig. 1, a block diagram illustrates a conversational browser system according to a preferred embodiment of the present invention. In general, a conversational browsing system 10 allows a user to access legacy information services and transactions through voice input (either uniquely or in conjunction with other modalities such as DTMF, keyboard, mouse, etc) using a standard networking protocol such as HTTP. In addition, it is to be understood that the HTTP protocol is a preferred embodiment of the present invention but other similar protocols can be used advantageously. For example, this can be deployed on top of any protocol that such as TCP/IP, WAP (Wireless Application Protocol), GSM, VoIP, etc., or any other protocol that supports IP (and therefore provide TCP/IP or similar features. Even more generally, if TCP/IP is not available we can implement another protocol offering features similar to TCP/IP or at least performing network and transport functions (the present invention is not dependent on the transport and network layer)..

In Fig. 1, a conversational browsing system 10 according to one embodiment of the present invention comprises a conversational browser 11 (conversational application) which executes on top of a CVM (conversational virtual machine) system 15. The conversational browser 11 comprises a transcoder module 11 which, in general, transcodes conventional (legacy) structured document formats such as HTML or DB2 into an intermediate document, or CML (conversational markup language) document in accordance with prespecified transcoding rules (as discussed below). A CML document describes the conversational UI of the legacy information format to be presented to the user. More specifically, a CML document comprises meta- information which is processed by a CML parser/ processor 14 to present, for example, HTML-based information to a user as synthesized audio messages. It is to be understood that various embodiments for a CML document are contemplated for implementation with the present invention. In a preferred embodiment described in detail below, a CML document is defined by tags which are based on XML (extensible markup language). It is to be understood, however, that any declarative method for implementing CML may be employed. XML is currently preferred because of its simplicity, power and current popularity.

The conversational browser 11 further comprises a command/ request processor 12 (a command and control interface) which converts user command (multi- modal) inputs such as speech commands, DTMF signals, and keyboard input into navigation requests such as HTTP

requests. It is to be understood that in a pure speech conversational browser, the only input is speech. However, the conversational browser 11 can be configured for multi-modal input.

When certain conversational functions or services are needed, the conversational browser 11 will make API calls to the CVM 15 requesting such services (as described below). For instance, when interpreting a CML document (via the CML parser/processor 14), the conversational browser 11 may hook to a TTS (text-to-speech syntheses) engine 26 (via the CVM 15) to provide synthesized speech output to the user. In addition, when speech commands or natural language queries (e.g., navigation requests) are input, the conversational browser 11 may hook to a speech recognition engine 24 and NLU (natural language understanding) engine 25 to process such input commands, thereby allowing the command/request processor to generate, e.g., the appropriate HTTP requests. The CVM system 15 is a shell that can run on top of any conventional OS (operating system) or RTOS (real-time operating system). A detailed discussion of the architecture and operation of the CVM system 15 is provided in the patent application IBM Docket No. YO999-111P, filed concurrently herewith, entitled "Conversational Computing Via Conversational Virtual Machine," which is commonly assigned and fully incorporated herein by reference.

Briefly, as shown in Fig. 1, the CVM system 15 comprises a conversational API layer 16 through which the conversational browser 11 can "talk" to a CVM kernel layer 17 to access (via system calls) certain conversational services and behaviors including the conversational engines 23. The CVM kernel 17 is responsible for allocating conversational resources such as engines and arguments (either local and/or distributed) and managing and controlling the dialog and context across applications and devices (locally and/or distributed) on the basis of their registered conversational capabilities and requirements to provide a universal and coordinated CUI (conversational user interface). The CVM system 15 performs conversational services and functions by implementing calls to local conversational engines 23, e.g., a speech recognition engine 24, a NLU (natural language understanding) engine 25 a TTS (text-to-speech) engine 26 (as well as other engines such as an NLG (natural language generation) engine, speaker recognition engine) through a conversational engine API layer 18 (such as SAPI, SRAPI, JSAPI, SVAPI or extensions of such engine APIs). In addition, engine calls can be made to remote speech engines in distributed topologies. In addition, calls to an audio subsystem 33 (providing audio capture, compression, decompression and reconstruction) and any other multi-modal I/O

resources (such as a keyboard 28, Mouse 29, touch screen, microphone/speaker 31 and DTMF 32) are performed via a conventional drivers/API layer 22.

The CVM system 15 further comprises a communication stack 19 for providing network communication via conventional protocols 20 such as TCP/IP, HTTP, WAP, etc. The communication stack 19 further comprises conversational protocols 21(or distributed conversational protocols) which are utilized for distributed applications. As described in the above-incorporated IBM Docket No. YO999-111P, the conversational protocols (or methods) 21 include protocols for (1) discovering network devices and applications that are "conversationally aware" (i.e., that speak conversational protocols); (2) registering conversational capabilities (resources) such as conversational engines and arguments between network devices; (3) negotiating network configurations (such as master/slave, peer-to-peer) based on registered conversational capabilities; (4) exchanging information to coordinate a conversation between network connected devices such as information regarding the state, context and history of a dialog, conversational arguments, applets, ActiveX components, procedural objects, and other executable code; and (5) speech coding protocols to transmit and receive compressed speech (waveforms or features). These conversational protocols 21, as well as their role for providing conversational coordination between networked devices is also disclosed in the patent application IBM Docket No. YO999-113P, filed concurrently herewith, entitled "System and Method For Providing Network Coordinated Conversational Services," which is commonly assigned and incorporated herein by reference.

It is to be understood that the engines 23 and I/O resources 27, conventional drivers/APIs 22 and audio subsystem 33 illustrated in Fig. 1 are components that are part of the underlying device, machine or platform on which the conversational browser 11 and CVM system 15 are executed. It is to be further understood that the conversational browser 11 and CVM system 15 can be provided as separate systems or, alternatively, the conversational browser 11 can be implemented as a stand-alone application carrying its own CVM system 15 (in which case the browser and CVM platform would be the same, i.e., indistinguishable entities). In addition, in the absence of a CVM system 15 as specifically described above, it is to be understood that the conversational browser 11implements functions that are similar to that discussed above for the CVM system 15 (e.g. the conversational browser would make API calls to appropriate engines locally and/or distributed). Indeed, the API, services, features, behaviors, access to engine and

communication mechanisms can all be built directly into, and made part of, the conversational browser application 11 as part of the features and services provided by the browser (this would be analogous as having a JVM (java virtual machine) provided by the underlying operating system to a Netscape browser or having the browser implement its own JVM). In addition, the conversational browser can take advantage of similar existing services provided by the underlying operating system and/or applications running in parallel of the browser.

It is to be understood that the transcoder 13 may be part of the conversational browser11 as illustrated in Fig. 1. Alternatively, as described below, the transcoder 13 may be located in a network and running, for example, on another machine, conversational proxy server, on a server of the content provider (server-side), or distributed on various machines in the network. In addition, the transcoder may be a proprietary application of the proxy server, browser or content provider. The transcoder should also be able to directly convert other proprietary formats like screen map of corporate software. The transcoder 13 comprises a functional transcoding module 13a for converting , e.g., an HTML document (or other format page) into a CML document. Various rules for implementing this conversion will be explained below. In addition, the transcoder 13 comprises a logical transcoding module 13b which accesses and interprets the logic information behind the application to build a meaningful dialog. This often requires in depth information about the application and /or business logic. It can be conveyed with extra XML or meta information that can define some of these intent. Another method which may be employed for performing logical transcoding is described in described in the patent application IBM Docket No. Y099-114, filed concurrently herewith, entitled "Structure Skeletons For Efficient Voice Navigation Through Generic Hierarchical Objects", which is commonly assigned and fully incorporated herein by reference. This application describes how conversational structure skeletons may be used to encode and transmit logical transcoding information.

Preferably, the transcoder 35 is a generic, general purpose converter, which is capable of transforming any transaction form or HTML page to a CML page. With respect to straightforward dialog like ATM dialog or IVR dialogs, wherein HTML dialog pages with form filling (HTML or CGI) or ATM dialog (and other 3270 type of screen dialogs) have limited information or provide limited user choices, the transcoding may be readily formulated in terms of CML pages. Other complex HTML documents can instead be tagged as CML in the first

instance on the server side (as explained below) without requiring a transcoder for the conversion. Indeed, as explained below, in a preferred embodiment, all the content provided to the conversational browser is directly written in CML without the need for a transcoder or transcoding proxy, as opposed to generic HTML pages, for instance, being transcoded on the fly by the browser or a server- side transcoder. Otherwise, basic composition and design rules can be imposed to simplify the conversion from HTML to CML. For instance, news and e-mail/voice-mail can be converted by filtering the incoming text with conventional TTS filters before tagging it with intonation and TTS markup information à la JSAPI.. In practice, these conversion rules can be either proprietary based on the service provider or the object of a standard. These rules may comprise the following:

1. If the HTML page comprises images, the transcoder will discard the images and generate the necessary meta information for presenting the image name and comment added (if any) in the image tag for audibly presenting such information.

2. If the HTML page comprises scripts and/or applets that have no component which may be presented audibly, the transcoder can generate meta- information for running special warning messages. The warning has several purposes. For instance, the warning message can inform the user that the HTML page contains a script or applet. Indeed in the absence of any speech output provided by the script or applet, the user may never know about the script or applet (as opposed to a GUI browser where an applet animation is immediately visible). In addition, the warning message serves to inform the user that he/she may be missing some information/content contained in the applet that is not capable into being transcoded into meaningful CUI (conversational user interface) features (this holds true for situations where the applet is written in conventional GUI languages like Java, instead of being written with the conversational foundation classes as part of CVM libraries (such as described in the above incorporated IBM Docket No. YO999-111P) to incorporate a CUI component.

In addition, meta-information ca be used to synchronize the GUI (visual) side of a multi- modal browser with the speech side, such as described in provisional application U.S. Serial No. 60/128,081, filed April 7, 1999, entitled "Multi-Modal Shell", which is commonly assigned and incorporated herein by reference. According to that invention, even procedural components (like applets and ActiveX components) can be transcoded into a GUI only or speech only (and variation of this) based on the capabilities of the device/browser. This automatically solves the

problem of the transcoding that was inherently coded by the application developer or the author of the components (foundation classes) the application used to build his procedure.

3. With HTML pages comprising frames, the transcoder can generate meta information for presenting the frames either by their name first or an index number. For instance, the audio playback may be "there are 4 frames on this page: frame 1 presents ....." All the frame links and functions created by HTML and CML can be activated as soon as the entire page is downloaded. This will allow the frames to be used as a menu. Alternatively, a rule can be applied such that only the main frame is presented, while the remaining frames are active but not read.

4. HTML pages can be filtered using TTS filters to produce JSAPI markup (or similar variations).

5. Any hyperlinks in an HTML page can be converted into baseforms, and FSG (finite state grammar). More specifically, hyperlinks and other commands and input that can be followed by the user (by voice commands) may be added to an active list of commands of the browser. This involves adding the vocabulary, the pronunciation rules of these words when needed (i.e the baseforms) and modifying appropriately the grammar or the NLU framework / context to support these new commands.

6. Text may be synthesized with different voices for different fonts. However, the conversational browser can elect not to change the voice.

7. Hyperlinks can be presented with an audio icon (sound, or a different voice, or a background music).

8. Each hyperlink can also be presented as a hidden menu, which may carry over to the following pages, if indicated, for example, with a <!--CML tag-->.

9. Pull down menus and constrained menus can be automatically converted to dialog with appropriate vocabulary and commands.

10. Free fields can activate NLU or dictation capabilities. In general, for general purpose dictation, the speech recognition engine used can be the local default engine of the conversational browser. Alternatively, for distributed systems (as discussed below), the content provider can provide a suggested URL of a speech recognition engine if the browser does not have the necessary capabilities or if the remote speech recognition engine is specifically tuned for the NLU or dictation application. NLU dialogs with NLU form filling can be performed by either passing the appropriate URL of the NLU engine as a CML comment tag in the HTML

page or by directly providing CML pages instead of HTML pages (as discussed in further detail below).

11. Multiple form based screen based transaction GUI dialogs can be directly converted into menus of sequential dialogs if each menu is a stand-alone component (e.g., performing a complete task such as obtaining an address or ordering an item)

The conversational browser 11 may also include a "barge-in" feature that allows the user to utter an attention word or otherwise interrupt the output flow of a TTS presentation before giving a new command or input (as explained in more detail below). This may be implemented with buttons to press when a person speaks, whereby the button is pressed to talk or pressed to start and/or press to stop.

Preferably, the CML documents generated by the transcoder (or the CML documents directly provided by the content provider as discussed below) comprise any combination of the following exemplary features and necessary tags to indicate:

1. Active links which are either spoken to, or hidden from, the user;

2. Links to conversational arguments (e.g., data files of vocabularies, grammars and baseforms) that may be used to decode possible input given by the user for the page;

3. For distributed systems, links to speech recognition URLs, speaker recognition URLs, TTS URLs and/or NLU engine URLs, etc., which may be used to decode user inputs and/or provide location(s) where to send a given page so as to generate the audio output. A local host may be used to impose a local (default) engine. Therefore, if no URL is given, the browser default engine is used. This may be set based on the browser, device, user preference or the service provider (ISP or ASP) preference. This amounts to using any relevant meta-information on the browser or on the server to set the browser default location;

4. URLs to audio files to play back to the user (e.g. IVR prompts);

5. Confirmation request tags that impose user confirmation to some input before following a given link;

6. DTMF code associated to a given link, as well as DTMF grammar for complex key entry;

7. TTS markup (à la JSAPI) describing how text should be played back to a user;

8. Scripting language and procedural code (such as Java, Javascript, ActiveX, etc.) so as to run conversational applications that direct their own dialog. Conversational programming

languages and foundation classes (i.e. CVM libraries) can be used when the browser is built on top of CVM (as discussed above). Otherwise, the code and services can be provided by the browser, the underlying operating system or other applications or services that are available in parallel with the browser.

9. Conversational capability tags, describing the browser minimum requirement for a resource (e.g. a script, an image, etc). For example, non-audio enabled scripts should not be downloaded or run on a speech only device, except if such scripts perform a meaningful task. In addition, images should not be downloaded on a speech only browser (although they are relevant on multimodal and GUI only browsers). When the presentation material is converted according the capabilities of the browser (i.e., the underlying engines), the entity that performs the conversion is a conversational proxy;

10. URLs of plug-ins to use/download for some specific processing (e.g. an encryption plug-in for speaker recognition stage or compression routine and front-end for a specific engine); and/or

11. Meta-information tag(s) for bookmarking and subsequent return to the page (e.g. set of keywords with their corresponding baseforms, grammars, etc.) which, when invoked, will propose the page on the list of the possible link to follow. This is typically a feature related to the categorization / meta-information service as provided by a CVM (as discussed in the above-incorporated IBM Docket No. YO999-111P).

It is to be appreciated that a specific comment **<!--CML proxy url=... -->** tag can be incorporated in an HTML page to specify a custom conversational proxy or transcoding to perform a desired transcoding function (conversational proxy, functional transcoding or logical transcoding). Alternatively, the conversion can be performed by a default transcoder (which is part of a browser, a server or a distributed proxy). CML pages may also be customized by the (proxy) transcoder based on the capabilities of the conversational browser. This functionality can be performed differently than by using the URL tag as a proxy (which implies that the browser connects to a proxy server and sends its HTTP requests through the proxy). If a CVM is present, an underlying handshake can occur to exchange meta-information about the conversational capabilities of the engines available to CVM for the browser, and then based on that the proxy can automatically

tailor the pages. Also an alternative consists of having in such case the browser forwarded to a different page suitable for the engine capabilities.

In addition, **<!--CML tag...-->** can introduce direct CML tags into an HTML or XML page, which will be directly processed by the conversational browser or conversational proxy. Each of these CML features will be discussed in more detail below.

While CVM is multi-modal, for specific browser/client capabilities (e.g. only a IVR), only the relevant functions are tagged. In the preferred embodiment, the browser typically ignores unknown tag instead of forcing a consistency of the ML. Furthermore, for general purpose systems, the browser should register its capabilities to the proxy or server, using a cookie mechanism or filling a hidden form when connecting for on the first page of that content provider or to the server gateway/portal. The cookie can carry the needed information to identify or authenticate a user. It can also carry all the context information about the current dialog status of the browser. This can also include the rest of the local context, history and/or preference information to be sent or shared with the server upon connection to it. The capability to send dialog information, context and other meta-information to a server via a cookie or other registration protocol enables the server to immediately participate in the conversation. Other registration mechanism may be implemented other than the cookie mechanism.

Conversational mark-up language rules can be added by the content provider to an HTML file or substitute to an HTML file to take full advantage of the conversational browser. This language should present tags to indicate the feature described in the previous section (as described below).

Telephony Applications Using Conversational Browser

Referring now to Fig. 2, a block diagram illustrates a conversational system for providing telephony services according to an embodiment of the present invention. The system 100 comprises an IVR 101 (or call center) which is accessible by a client device 102 over a telephone network 104 (e.g., PSTN (public switched telephone network)). The IVR 101 is connected to one or more content servers 105 over network 104 . In this embodiment, it is assumed that the IVR 101 runs on a machine on the premises of a telephone company (telco). The IVR 101 may be, for example, associated with a service provider of a banking application to provide ATM telephony services. The client device 10 may be, for example, a conventional telephone, a

mobile telephone, or a PC having a telephony card, to establish communication over the telephone network 103 to the IVR 101. The network 104 may be the Internet, an intranet, a LAN, *ad hoc* networks, etc. The content servers 105 may be HTTP servers, for example, which comprise the content pages (HTML, CML, etc) that are associated with the IVR 101 ATM services.

The IVR 101 comprises a conversational browser 106 which is illustrated as a stand alone application running a CVM 106a. It is to be understood that the conversational browser 106 and CVM 106a of Fig. 2 are similar in architecture and functionality as the conversational browser 11 and CVM 15 described above with respect to Fig. 1 (whereby the CVM can be included within the browser or in an underlying operating system. The IVR system100 further comprises an audio system 107 providing audio functions such as audio capture, acoustic front end (acoustic processing) and an audio subsystem (e.g., soundcard). The IVR 101 further comprises a DTMF processor 108 and conversational engines 109 (e.g., speech , NLU, TTS engines, etc.).

When the client 102 is connected to the IVR 101, input commands such as speech commands and DTMF signals are received by the acoustic front end of the audio system 107 and sent to the appropriate conversational engine 109 (e.g., speech engine) and DTMF processor 108, respectively. The processed speech/DTMF commands are then passed as calls to appropriate functions in the conversational browser 106. The conversational browser 106, in turn, generates appropriate requests (e..g, HTTP requests) to obtain desired information from one or more of the content servers 105 over the network 104. The content server 105 retrieves the appropriate information (or completes the specified action) and then transmits results, messages and/or menu options (using TCP/IP or similar protocols discussed above) in either HTML (or other similar formats), CML, compressed audio, or a combination thereof (depending on the manner in which the information is structured on the content server 105). As discussed above, for HTML (and other structure documents), the conversational browser will transcode the information (via a transcoder 13 discussed above) to generate a CML document for processing (via the CML parser/processor 14 discussed above), whereas CML documents from the servers 105 can be directly processed (without transcoding). The conversational browser 106 can hook to necessary conversational engines (via the CVM 106a) to establish a dialog with the user. As discussed below, the conversational engines may be distributed over the network 104.

The conversational browser can also decode compressed audio which transmitted from the content server 105 using a standard network protocol (such as HTTP), and then play back the audio to the user as telephone audio. (thereby allowing, e.g., IP telephony via the conversational browser 106). Conversely, audio input can be captured via the audio capture portion of the audio system 107 and transmitted (via the conversational browser 106) in compressed or non-compressed format to a server over the network 104 for distributed recognition and processing (such as described in the above incorporated patent application IBM Docket No. YO999-113P). As indicted above, the transmission of the compressed speech is performed by the communication stack of the CVM 106a (such as discussed in YO999-111P, as well as YO999-113P). In the absence of a CVM, other communication engines will perform the handshake , communication and coding/decoding. It is to be understood that such communication is not limited to HTTP or VoIP, it can involve any TCP/IP and related protocols. For example, WAP, Bluetooth, Hopping communication protocols may be employed with the present invention. When the audio is captured locally and shipped to a remote server, this can be performed using a plug-in (native implementation) or for example with a java applet or java program using audio and multimedia API to capture the user's input. In addition, the conversational communication(coding) protocols may be used for transmitting compressed speech using, for example, socket connections, RPC, RMI, etc. (such as described in the above incorporated patent applications IBM Docket Nos. YO999-111P and YO999-113P).

It is to be appreciated that the system described in Fig. 2 is only one possible embodiment for providing IVR services. In another embodiment, for instance, the IVR can behave solely as "soundcard" of the system in the sense that it provides the audio capture functionality to capture input speech from a user and then transmit the captured speech to a web server having a conversational browser (similarly it receives audio output from the server and transmits it to the IVR for play back to the user). In this case, the IVR acts as a soundcard or an audio subsystem for the conversational browser. Alternatively, the IVR may be providing the conversational engines (speech recognition engine, TTS engine, speaker recognition engine etc...) as well as the audio capture and all the system management, call balancing, load balancing functions etc. and only use the conversation browser as a driver of the dialog. Alternatively, also the IVR can only performs the audio capture and the

load balancing/system management while having the conversational engines running on other servers and the conversational browser running also on other servers. Eventually there is a possibility that the IVR or call center is implemented with a farm of networks small workstations (typically NT machines) each with a telephony card that will own the audio capture, the conversational engines and the conversational browser associated to one or a very limited of active ports (i.e active calls).

In the embodiment illustrated in Fig. 2, it is assumed that all the functions of playing back prompts, synthesizing speech, generating dialog, understanding DTMF input or speech input are performed at the level of the IVR 101. It is to be understood that in the absence of a CVM 106a, it is the responsibility of the system integrator or IVR vendor to provide a concrete implementation a system providing abstract layers similar to those provided by the CVM (as discussed above). The actual implementation of the system will determine the actual architecture of the IVR 101, but such architecture would be independent with respect to the functionality and operation of the conversational browser 106.

Advantageously, the conversational browser 106 effectively changes the IVR 101 from what is known in the art as the "all-time" connection to a packet switched network on the service side. In addition, with the universal IVR architecture employing a conversational browser, it is to be appreciated that the content pages are on the same HTTP servers and use the same information access methods, sharing the legacy database access SW layer, etc. as web servers for information access. In other words, an IVR can now be considered a special case of a HTTP server with a conversational browser. Depending on the device, server and content/service provider capabilities, the conversational browser can be located on the client, on the server, shared between the client and the server or distributed across the network.

Referring now to Fig. 3, a block diagram illustrates a conversational system for proving telephony services according to another exemplary embodiment of the present invention. In this example, the conversational browser 106 is located entirely on the client device. All the processing performed by the IVR (as explained above with reference to Fig. 2) is now performed locally on the client device and only the fetching of HTML documents, for example, is performed through HTTP on the content servers 105. The content servers 105 can be HTTP servers having applications executed thereon which are associated with IVR applications. Alternatively, the system of Fig. 3 can simply be used for browsing the WWW.

For instance, as illustrated in Fig. 3, the client device can be a PC 60 (personal computer) having a conversational browser 106 (and CVM 106a) and conventional telephony platform 61 (comprising, for example, a modem, soundboard, speakerphone and voicemail system) for providing telephony services via the PC 60. The PC 60 can be used to establish a dial-up connection via modem (through the conversational browser 106) over the telephone lines 102 to the telco or any other remote access server (RAS) 70 (e.g., Internet Service Provider (ISP)) to access content servers 105 over the network 40 (e.g., Internet, intranet). The PC 60 can also access local/private resources or the Internet through remote connections via IP connection using a dedicated line L. The PC 60 with the browser 106 can be used to provide conversational browsing of, e.g., the WWW. As explained above, the conversational browser 106 performs, e.g., the recognition of voice commands and the conversion of the voice messages from HTTP server coming compressed over Internet. The conversational browser 106 can access any speech enabled local applications 62 (calendar, etc) running on the PC 60 or local client.

The client device may be a mobile phone 80 (e.g., a GSM terminal) having suitable architecture and CPU power to run the conversational browser 106 locally. The mobile telephone 80 can download DSP code (or any other type of CPU programming language) to interpret the Internet protocols for speech (Internet telephone) (or other protocols such as WAP) as well as compression algorithms for speech purposes and any other application or speech processing programs written, for example, in some derivation of Java. It is to be understood that the DSP code may be native and preloaded on the phone (especially if an instantiation of CVM 106a is present on the device). Furthermore, even higher level mobile phone with screen can have higher level of locally running browser with text capabilities or even display capabilities. In addition, the new telephones can be equipped with higher sampling rate A/D converters. This would enable to take advantage of the cepstrum coefficient compression to send the speech more effectively over the communication lines and increase the speech recognition quality (for distributed Speech recognition such as explained in the above-incorporated IBM Docket No. YO999-113P).

It is to be appreciated that the functions of the conversational browser 106 may be partially located on a local client device and partially located on a remote server. For example, the acoustic features can be computed by the client device (e.g., via the front end module 107, Fig. 2)), compressed and then transmitted to the server at low bit rate for processing. In

addition, a portion of the browser processing can be done locally, e.g., playing back regular text and getting simple input, and some processing can be performed on the server side, e.g., obtaining street addresses or complicated input or ascii input to forms. It is to be further appreciated that the browser functions can be distributed across a client device, one or more servers, and a network of resources. For instance, the browser can be located on the server, the client or shared between the two, with conversational subsystems (e.g., speech recognition engine, NLU engine, TTS) being located at other areas on the network. It is to be appreciated that even the browsing functions can be shared between networked devices (servers, computers, PDAs etc). Again, a CVM is a key element to "manage" such cases through communication protocols similar to those described in the incorporated patent applications IBM Docket Nos. YO999-111P and YO999-113P. In addition, for domain specific applications, a content provider can directly provide its own TTS, NLU or speech recognition engine.

Advantageously, the deployment of the present invention can afford the preservation of legacy applications that provide information pages in HTML format, by allowing such information to be presented for pure conversational interaction (besides the capability to display the pages or portions of the pages with multi-modal devices). Each of these concepts will now be illustrated with reference to the illustrative embodiments of Figs. 4a and 4b.

Referring now to Fig. 4a, a block diagram illustrates a distributed system wherein the resources for providing conversational browsing and IVR service are shared across one or more networks. It is to be understood that the dotted circles in Fig. 4a represent, for example, different locations of a given network or individual networks (or subnetworks) on an intranet or the Internet, each comprising corresponding servers, hosts, which interact with one another via standard network protocols (e.g., TCP/IP, HTTP). A PC 302 having a GUI terminal 300 and conversational browser 302a can be used to provide a combination of GUI and conversational browsing (it is to be understood that the device 302 can be any client device (other than a PC) such as a phone which is connected the distributed network via a PSTN , or a bluetooth device connected through bluetooth to a TCP/IP network, a setopbox connected through ISDN or cable or satellite link to the distributed network). Alternatively, if the PC 302 can access a remote server 303 having a conversational browser a via IP connection. Likewise a user can dial into an IVR server/PC 302 via telephone 301 (e.g., regular, cellular, or smart phone) obtain perform conversational browsing or IVR via the local browser 302a or via the remote server 303 and

conversational browser 303a. Alternatively, as explained above, the phone 301, PC/IVR server 302 and remote server 303 can share the functions of the conversational browser depending on their respective architecture and processing power capabilities. In addition, the conversational browser 303a can be shared across one or more servers 304 distributed across a given network or subnetwork. Alternatively, each of the remote server 304 can support one or more specific functions of the browser 304a. Depending on the respective conversational browser function, the PC/IVR server 302 or remote severs 303, 304, each may hook (as necessary) via TCP/IP, HTTP, API (for CVM instantiations using the communication protocols discussed above) to one or more conversational subsystems 309 distributed over the network. In addition, any required conversational arguments 308 can be accessed for use by the conversational browsers 302a, 303a and/or 304a or conversational subsystems 309 over the network.

The exemplary system further includes a conversational proxy server 305 with transcoder 305a, which may be accessed over the network to convert HTML documents and other applications to CML in circumstances where the conversational browsers 302a, 303a, 304a cannot perform such conversion due to inadequate processing power of the device on which the browser runs. In addition, the conversational proxy and transcoder 305a may be associated with a content service provider to perform conversion of HTML documents in instances when the conversational browsers 302a, 303a and/or 304a lack specific proprietary information about the application to perform such conversion (which is known only by the content provider). As shown in Fig. 4a, the HTML documents are generated via web server application of the content provider using a traditional transcoder 306 that interacts with database 307 via SQL or a proprietary API to convert proprietary information into HTML form. It is to be appreciated that the communication between elements 306 and 305/305a can involve the transfer of XML or meta-information to assist the conversational transcoding/proxy task. In addition, structure skeletons can be used as described in the above-incorporated application Y0999-114.

Fig. 4a illustrates a voice portal, which is a CML server where transactions allow the search of HTML or CML pages or web sites, which is similar to today's web portals. The voice portal can retrieve desired sites and pages. These CML pages are directly presented to the conversational browser. Pages retrieved in another presentation language (e.g., HTML) are transcoded before being transmitted to the browser. The main difference between a voice and web portal (i.e., HTML or WML (wireless markup language) based) is that the portal downloads

the requested pages and transcodes them appropriately before sending them to the browser. For client devices (or browsers) having transcoders, the pages can be send to the browser in another format (HTML and \/or HTML-based structure skeleton) and transcoded on the fly by the browser into CML as an intermediate internal language. The voice portal service can be subscription based, paid by the carrier/wireless provider, paid by the referred content provider (on a hit basis), on complete profitable transaction basis, on a percentage of profit from referred business, on a subscription basis to have the service present in the search engine, or the service ranking high as result of the search engine, or for quality CML pages (hand prepared or reviewed) as compared to the automated approach.

Referring now to Fig. 4b, a diagram illustrates an exemplary embodiment of the present invention for providing conversational browsing or IVR services in conjunction with legacy corporate software applications. The system of Fig. 4b is similar to the system described above (Fig. 4a) except that Fig. 4b specifically illustrates conversational browsing of a conventional 3-tier (distributed) corporate application on a given corporate network. The 3-tier application includes a front end terminal/presentation server 310 (Tier 1) which contains the programming to create and present HTML-based documents, a middle tier business logic server 311 or application server (which contains the business applications and serves client requests from workstations in a LAN, for example), and a back-end or third tier 312, 313, which comprises the legacy database and transaction management (data control) applications. As shown, conventional presentation is accomplished via GUI 314, telephone (IVR) 315 and 3270 type screen dialogs (3270 terminal are types of terminals that are typically supported by mainframes and CISC systems, e.g., the traditional greenish terminals that may be found at ATMs which are mostly ascii character only-based screen). Straightforward HTML dialog pages such as form filling (HTML or CGI) or ATM dialog (and other 3270 type of screen dialogs) or other dummy terminals (CT 100, Palm Pilot screen, etc.) can be readily converted to CML via transcoder 305a of proxy server 305, for example, since the business logic of such transactions is typically driven by simple dialog so that limited amount of information or choices is presented at once to the user. The choice or menu can also be conveyed through skeletons as described in the above incorporated IBM Docket No. YO999-114P. In addition, transcoding can be readily extended to other proprietary APIs or screen maps **(e.g.,** maps of the 3270 screens or other screens into an HTML page usually performed by mapping programs called screen scrapers). In particular,

since programs currently exist for transcoding screen maps to HTML pages, and screen maps are limited to simple dialogs driving the business logic, the conversion of the resulting HTML pages is a trivial case of the conversational proxy.

Advantageously, each of the above-described exemplary system architectures advantageously allow universal access via pure speech interface (conversational browser), pure GUI, and a combination of speech and GUI to the same bank of transaction and information services through a same protocol without requiring any redesign of the content currently provided over HTML and other XML/HTTP protocol.

Conversational Applications Employing CML

The following discussion of exemplary embodiments describe network- based conversational applications using CML (conversational markup language) in accordance with the present invention. The term "conversational application" used herein refers to an application that interacts with the user via voice input/output. The term "network-based conversational application" used herein refers to one in which the elements of the conversion that define spoken output and input, i.e., pages of CML, may be obtained over a network or locally. Indeed, just as GUI applications can be built using a set of interlinked pages written using HTML markup language, conversational systems can be built using a set of interlinked pages written in conversational markup language - CML. Moreover, just as HTML provides a set of mechanisms for translating GUI actions into application actions such as visiting other pages or communicating with a server, CML similarly can be implemented to provide mechanisms for translating spoken inputs into application actions. The term application in this context should be understood in the same broad sense in which a set of related HTML pages containing text, links, and forms, together with the code on a server which processes the forms (e.g., servlets or CGI scripts), constitute an application. In addition, similar to the manner in which procedures (such as conversational foundation classes or other dialog components) can also be loaded via Java or ActiveX components, conversational applets and foundation classes can be loaded /downloaded to execute procedural conversational actions (which is a feature of CVM).

Referring now to Fig. 5, a diagram illustrates an exemplary network-based conversational system comprising various classes of conversational applications enabled by CML according to the present invention. The illustrative system comprises a client device 500 having a

conversational browser 501. The client device 500 may be, for example, a desktop PC, a PDA, an automobile computer, a smart telephone, or a "dumb" telephone (the term "dumb" telephone refers to herein as a device that functions as a microphone at the end of a long cord and needs to be connected to a browser at the other end of the phone line). The client device 500 may also comprise one or more speech- enabled local applications 502 running on the client 500 using CML to communicate with the conversational browser 501. For example, the local application may be a car navigation application in which a "Speech Navigation Application" interacts with computer mapping software and a GPS (Global Positioning System) device to provide conversational driving directions. Similarly, other applications and services provided locally by the underlying conventional OS, or other applications can be used to provide services (such as disambiguation, context access, etc.) and dialog modules to the browser when there is no underlying CVM to provide such services.

The client device 500 can access any one of a plurality of server systems S1, S2, and S3 over network 503 (e.g., the Internet) using a standard network protocol (e.g., HTTP, TCP/IP) to transmit CML pages to the client device 500, which are processed by the conversational browser 501 to provide a conversational UI to the user of the device 500. For example, the client device 500 can connect to server S1 to access existing HTML information via a transcoding proxy which, as described above, is capable of interpreting HTML, extracting relevant information, and generating CML information. In addition, the client device 500 can connect to a specialized web server application (S2) such as Lotus Domino server to access Notes data (e.g., Notes e-mail) via a CGI application. In particular, the Domino server can be configured to generating CML and transmit the CML using HTTP. In another example, the client device 500 can connect to a web server application (S3), using a CGI application or Java Servlet to access an legacy database of an enterprise, wherein the web application generates and transmits the information in CML.

This lead to another interesting application example: a CML based universal messaging system. In particular, the same way that conventional and currently available mail and messaging systems can be provided with HTML by ISP, ASP (Application service providers) and web portals, a voice portal or voice web service provider can use CML to offer e-mail access and processing and universal messaging. Universal messaging, means that you can now access via voice different messaging services and program paging, e-mail, voicemail and call processing by

voice. Because the interface is now built in CML through the conversational browser, the user can process his/her messages automatically. Similarly call transfer and processing can be seen as a combination of a CML interface announcing the caller information and possibly call topics and offering a menu of options. The user can select actions (transfer, store, take the call). Each of these actions are processed as procedural call to the call processing unit and the call is accordingly processed (e.g. forwarded or transferred to the user). With a pure speech conversational browser such as shown in Fig. 5, speech and audio are the only output provided to the user. When a user desires, a page is downloaded from the service provider (in CML). The CML is used to describe the conversational UI to be presented to the user. Using the TTS markup information, the conversational browser will read the page to the user and listen for commands from the user to be decoded by the speech recognizer (using possibly clues from the speech recognition portion of the CML page) to follow links, fill forms, or command the browser. For non-embedded applications, the conversational browser preferably utilizes a large vocabulary continuous speech recognition system (e.g., IBM Via Voice) and a "Free text rule-based TTS using JSAPI (java speech) - including dynamic grammar compilation and NL process for example as described in the reference by K.A. Kishore, et al. entitled "Free-flow dialog Management Using Forms," Proc Eurospeech 1999, Budapest Hungary, September 1999 and Davies et al., "The IBM Conversational Telephony System For Financial Applications," Proc. Eurospeech 99, Budapest Hungary, September 1999>

Advantageously, the above capabilities such as server-based CGI scripts and Java Servlets, and local plugins, can provide mechanisms for application-specific code to participate in the conversational application. The interface between the application code and the conversational browser 501 can take the form of URLs for requests and markup data stream (HTML or CML) or sets of attribute value n-uples. Note that instead of URL, other addressing schemes (e.g., sockets and ports) to access local or distributed services may be used.

It is to be appreciated that each of the preceding IVR and conversational browsing applications could alternatively be built with application code with the applications communicating via API and remote calls (i.e., not implementing a conversational browser and CML). In particular, such applications can be written directly to a speech API such as the Java Speech API (JSAPI). However, for distributed applications, the currently available engine APIs (SAPI, JSAPI) have not been designed with network considerations in mind. Therefore,

networked/distributed speech applications directly written on top of the conversational engine APIs will always require re-implementation of the distributed functionalities. It is to be appreciated, however, for applications written on top of the CVM system (IBM Docket No. YO999- 111P), these limitations of the current engine APIs are virtually eliminated through the conversational application APIs that are communicated via the communication stack which hides most of the complexity of the engine APIs through the services and behaviors offered by the CVM shell. Indeed, complex dialog tasks can be efficiently performed though procedural implementations. Such procedures allow simple use of macros for such complex tasks. In addition, procedures are inherently expected to be more efficient than interpreted declarative language such as XML. Depending on the context of use (the importance of the efficiency vs. programming, portability across platforms, etc) , however, both alternative (declarative and procedural) may be used.

CML and the conversational browser, on the other hand, advantageously allow distributed conversational systems to be implemented which take advantage of legacy applications that have already been develop using HTML, for example. Unlike the currently available speech APIs, CML can readily support networked conversational applications. The conversational browser can interpret streams of CML originating from multiple locations thereby providing a user with the same kind of seamless experience provided by a desktop windowing system or by a visual browser. For example, while completing an airline ticket purchase the user can temporarily suspend the transaction and interact with a banking application on a different server to check an account balance. This is related to the concept noted above of having an appropriate mechanism to keep and transmit/restore the context of a dialog with the browser and possibly the server side processing. Furthermore, CML provides a higher-level tool that simplifies the construction of conversational applications (similar to the manner in which HTML simplifies the building of visual applications). A conversational markup language also reduces the amount of expertise in speech required. In addition, as stated above, building distributed applications on heterogenous platforms using an currently available speech APIs is difficult. The large-scale deployment of distributed GUI applications on heterogeneous platforms via the WWW demonstrates the feasibility of a CML data stream approach.

Referring to Fig. 6, a block diagram illustrates a general distributed conversational browsing system using CML in accordance with an embodiment of the present invention. In

general, a distributed conversational browsing system comprises an information server 600, a presentation server 601, a speech server 602, speech models 603 and a conversational browser 604. The information server 600 represents, for example, enterprise databases, newswire feeds and/or WWW servers. The information server 600 provides data in an "information format" which represents application-specific information in a device-independent manner. Examples of this data format include APIs for data access (e.g., Notes DB API, SQL) and application-specific data streams (news wire feeds, information marked up with XML content tags).

The presentation server 601 retrieves information from the information server 600 using an API or a protocol defined by the information server 600. The presentation server 601 then transcodes the received "information format" to a "presentation format" for presentation to a user. The "presentation format" represents information in a device-specific, application-independent manner for presentation to a user which, in accordance with the present invention, is CML (as described in further detail below). Conventional "presentation formats" include, for example, APIs such as the Windows GUI, API and data streams such as HTML, HDML (Handheld-device Markup Language) and WAP (wireless application protocol). The presentation server 601 can be, for example, an HTML server, CGI scripts and Java Servlets, Domino as an HTML server, and the Notes Client, which are configured for transcoding the information received from information server 600 into CML. The CML generated by the presentation server 601 is used to describe the conversational UI to be presented to the user via a conversational browser 604 and provides the mechanisms for translating spoken inputs into application actions.

The speech server 602 comprises the engines which are responsible for speech recognition and parsing and other conversational engines. The speech server 602 addresses the problem unique to speech: the speech recognition process may require large application-specific data sets (i.e., speech models 603) which it is not feasible to transmit to the client. This implies that the speech recognition process must be carried out where the language models reside near the presentation server 601 (in a transmission bandwidth sense). A standard network protocol such as HTTP, XML, VoIP, WAP protocol, as well as the conversational speech coding protocols describe in the above-incorporated YO999- 113P application may be used for transmitting audio from the conversational browser 604 to the speech server 602 and returning

the parsed results, as well as transmitting audio, such as recorded or synthesized speech, from the speech server 602 to the conversational browser 604.

Referring now to Fig. 7, a diagram illustrates an exemplary distributed conversational system using CML in accordance with an embodiment of the present invention. More specifically, the exemplary embodiment illustrates a speech only client device 606 having a conversational browser 604 and microphone/speaker 605. The client device 606 represents, for example, a speech-only PDA (Personal Digital Assistant) or PVA (Personal Vehicle Assistant) client device connected, for example, by a wireless link to a network 607 (e.g., the Internet). The information server 600, presentation server 601 and speech server 602 (with speech models 603) together provide an application (e.g., weather information) to the client device using CML. The dotted line around the presentation server 601, speech server 602 and speech models 603 indicates that these elements are tightly coupled, in that the speech server 602 has speech models specific to the application. In addition, the presentation server provides a GUI to other browsers 608 using HTML. It is to be understood that the conversational browser 604 may access other conversational applications over the network 607.

Referring now to Fig. 8, a diagram illustrates another exemplary distributed conversational system wherein the client device 606 is a speech and GUI client. The client device 606 comprises a display screen 609, a microphone speaker 605 and a conversational + GUI browser 604a for presenting information to a user using speech, GUI and a combination thereof. In this embodiment, the presentation server 601 can provide both HTML and CML documents depending the request by the client 606. In addition, this system can be implemented by embedding CML markup can be embedded in HTML markup and extending a standard HTML browser (using a supplied browser extension mechanism) to run the conversational browser alongside the standard browser to provide a conversational interface alongside the GUI interface. A system and method for integrating and coordinating between A GUI and speech UI is explained in the provisional application U.S. Serial No. 60/128,081, filed April 7, 1999, entitled "Multi- Modal Shell", which is commonly assigned and incorporated herein by reference

Referring now to Fig. 9, another distributed conversational system using distributed telephony is shown. In this exemplary embodiment, a user can dial into a telephony platform 610 of the client 606 (such as a PC) to be connected to a conversational browser 604, which gives the user access to a selection of speech applications via the network 607 (e.g., Internet).

Alternatively, a user can connect to the speech server via dial in or directly via another existing connection (LAN, Bluetooth,, DSL etc...). In this case, the conversational coding algorithm presented earlier can be used to ship the data to the browser. Note that the client "device" is now essentially a telephone handset coupled to the conversational browser 604 running on the telephony platform 610.

Next, in the exemplary embodiment of Fig. 10, the user will dial into a given phone number to access one application. The system is a simplified version of the distributed telephony system of Fig. 9, wherein the conversational browser 604 and application are the same. In this embodiment, however, the conversational browser 604 is not connected to a network but can access only the specific application. The dotted line 611 in this embodiment emphasizes that the presentation server 601, the speech server 602 (and models 603), the conversational browser 604, and the telephony platform 610 are all operated as an integrated service. This is essentially a traditional telephony application, with the conversational browser 604 serving as dialog engine.

Conversational Markup Language: CML

This following describes a preferred embodiment CML which may be employed in the conversational browsing applications described above. In a preferred embodiment, CML is an application of the Extensible Markup Language (XML). XML is an emerging and popular web standard for HTML-style markup languages, defined by W3C, the same body that maintains the HTML standard. XML is essentially a generalization of HTML which borrows a number of design points from HTML. More generically, XML is a declarative universal format for structured documents and data. Advantageously, the use of XML as a basis for CML allows CML to be readily embedded in HTML or combined with HTML to create multi-modal applications (e.g., speech and GUI). It also allows JSML (Java Synthesis Markup Language), another XML application, to be embedded in CML and used as the speech synthesis markup language for CML. Advantageously, the standardization and popularity of XML make it probable that authoring tools for creating CML pages, as well as programming tools for the server code to generating CML, will become standard, therefore providing open architecture for the conversational browsing systems described herein.

XML comprises markup symbols to describe the contents of a page or file. The XML entities comprises pairs of tags of the form:

<**tag** *attr1*="val2" *attr2*="val2"> *arbitrary text*</**tag**>.

The extensibility of XML arises from the fact that the markup symbols of the XML application (and CML in this case) are unlimited and self-defining (i.e., the programmer is free to define his/her own tag names, attribute names and value sets).

In one embodiment of the present invention, CML is written in JAVA and a conversational browser utilizes a "what you hear is what you can say" style of speech input for purposes of local speech recognition and parsing. In other embodiments, the CML is extended for NLU speech input, and distributed client-server speech applications, wherein digitized speech is sent from the client via HTTP, for example, to a speech server and the parsed results are returned using CML (as noted above). Again, besides using HTTP or other conventional protocols, it is to be understood that speech can be transmitted to the speech server via the conversational communication protocols as discussed in the above-incorporated patent applications IBM Docket Nos. YO999-111P and YO999-113P. These embodiments will be discussed in detail below.

In general, CML is preferably defined as a set of tags, wherein the primary CML elements are **<page>**, **<body>**, **<menu>**, and **<form>**. In general, a "page" element groups other CML elements, and serves as the top-level element for a CML document. A **<body>** element specifies output to be spoken by the conversational browser. A **<menu>** element presents the user with a list of choices, and associates with each choice with a target, e.g., a URL, identifying a CML element to visit if the user selects that choice. A **<form>** elements allows the user to provide one or more pieces of information, where the content of each piece of information may be described by a grammar. The **<form>** element may also specify a URL to visit when the user has completed the form. The phrase "visiting an element" (or "visiting a CML element") refers to an action taken by the conversational browser, typically in response to a spoken user input (although it may also be in response to some other kind of user input or some other kind of asynchronous event). Visiting a CML element will cause the conversational browser to produce a spoken output, depending on the type and content of the CML element visited. For example, a **<body>** element is read, the choices of a **<menu>** element are listed, and so on. Visiting an element can also affect the set of spoken response that the browser will

accept from the user, as described for each element in the following sections and as discussed more fully in the section on dialog issues.

The following describes in further detail the tags (elements) and attributes that comprise a CML document or file according to a preferred embodiment.

Page Elements: A **<page>** element is the top-level CML document element that comprises one or more related nested CML pages or units (a unit is a **<body>**, **<menu>**, or **<form>** element) and is preferably structured as follows:

<page ttl="seconds"> (bodies, menus, and/or forms units) or nested pages **</page>**

The attribute ttl ("time-to-live") specifies the number of seconds that the CML page may be stored in a cache, wherein a value of 0 prevents caching of the page. A **<page>** element per se does not contribute (no spoken output or input) to the conversation between the conversational browser application and the user, but serves as a container to group other CML elements (e.g., menu, body, and form elements). A **<page>** element may also contain nested pages. The function of the **<page>** element as a (nested) container will be explained later. Visiting a **<page>** element is equivalent to visiting the first unit in the page. The nested page can also include conversational dialog objects or foundation classes. This can be implemented procedurally or with some equivalent CML pages. When a page is reloaded, a cookie or appropriate protocol/service/API is provided to reload the previous context.

Menu Elements: A **<menu>** element serves a function for CML which is analogous to the function that hypertext link and menu serves for HTML in that it presents the user with a set of choices. In general, a **<menu>** element specifies an introductory message and a menu of prompts to be spoken when the entity is visited, and a corresponding set of possible responses, and for each choice a URL, for example, to visit if the user selects that choice:

<menu>
introductory message text
<choice target=*"URL1"***>***prompt text 1***</choice>**
<choice target=*"URL2"***>***prompt text 2***</choice>**
</menu>

When a **<menu>** element is visited, its title text (introductory text) is spoken followed by the prompt text of any contained **<choice>** elements. A grammar for matching the title text of the

<**menu**> may be activated when the <**menu**> is loaded, and remains active thereafter. When the user utters a word or phrase which matches the title text of a <**menu**> element, that <**menu**> element is visited.

Each <**choice**> has a target URL that the conversational browser visits if the user selects that choice:

<**choice** target="target">*prompt text*</**choice**>

A target may be any valid URL (including relative URLs and URLs specifying HTTP:, file:, FTP:, and local: protocols (see below for a description of the local: protocol) or a URL plus a reference (using the standard URL#reference syntax.). Any other addresses, schemes, and protocols may be employed. The reference (i.e., the part of a URL after the #) is a string that names the unit within the document that has a name attribute whose value is the reference string. A URL without a reference is considered to refer to the first unit within the document. A relative URL is resolved relative to the URL of the containing document.

Furthermore, a "target" may be another address such as a socket address (IP address and port ID). In such case, other IP protocols such as the conversational protocols described in the above-incorporated patent applications IBM Docket No. YO999-111P and YO999-113Pcan be implemented. In situations where a specific non-conventional protocol is used, an additional argument can be used to activate communication such as the following:

<**choice target**="target" **Protocol**="protocolidentifier"> </**choice**>

This provides a mechanism to ship the context and meta-information. It also provides a way to provide system calls to CVM services or, in the absence of CVM, to present calls to equivalent services implemented as applications on top of the underlying OS, or within the browser platform itself. If the browser supports the protocol, the identifier is enough to activate appropriate communication. This is especially applicable for a conversational browser built on top

of CVM (as described above and in IBM Docket No. YO999-111P). In other cases, the protocol identifier can also point to a URL (plug-in or applet) to download so as to initiate the communication. It can also directly point to a local executable on the client platform supporting the conversational browser. Typical examples are "choices" that require distributed recognition (e.g., where the recognition is performed on a remote networked server) (as described in IBM Docket No. YO999-113P). Other examples include choices that download procedural functions

(or local functions) which implement the dialog. Again these conversational procedures are built on top of CVM. It is to be understood that these concepts are valid for all "targets" and not just "choice."

The "prompt text" of a **<choice>** element is spoken when the containing **<menu>** is visited. A grammar for matching the prompt text of a **<choice>** element is activated when the containing **<menu>** is loaded, and remains active thereafter. When the user utters a word or phrase that matches the prompt text of a **<choice>** element, the specified target is visited. For instance, the user may select one of those choices by saying an attention word or phrase ("computer", "go to" or "select"), followed by one or more significant words from the choice. The following example will illustrate the menu element:

```
<menu>
    Please choose from the main menu.
    <choice target="file:e-mail">E-mail.</choice>
    <choice target="file:news">News.</choice>
    <choice target="file:nav">Navigation.</choice>
    <choice target="file:mcform">Food Ordering.</choice>
    <choice target="file:weather">Weather information.</choice>
    <choice target="file:tutorial">Tutorial.</choice>
</menu>
```

The main menu may serve as the top-level menu which is heard first when the user initiates a session with a conversational browser. More specifically, when the conversational browser visits this menu it produces the spoken output "Please choose from the main menu" followed by a list of choices: "E-mail. News. Navigation. Food Ordering. Weather Information. Tutorial." Once the conversational browser has loaded this menu, the user may activate (select) any of the choices (during the remainder of the session) by speaking a command. The allowable commands will depend on the input technology being used. In one embodiment which implements a "what you hear is what you can say" approach, the allowable commands may comprise attention phrases (such as "go to" or "select") followed by a subsequence of the words of the prompt text, e.g., "go to e-mail" and "select news."

After the user has uttered a selection, the conversational browser will visit the target address (e.g., URL, socket address) specified by the target attribute associated with the given choice (e.g., the contents of a target URL are fetched and interpreted as a CML entity, and that entity is visited). Note that in the above example menu, all of the target URLs are relative to the URL of the containing page, which in this case is a file: URL that refers to the demo\main file - for example, the target "file:news" is interpreted as a file called "news" in the same directory as the file that contains the main menu.

In summary, the user may speak a phrase that matches the title text of a **<menu>** element to visit the **<menu>** element. The user may speak a phrase that matches the text of a **<choice>** element to visit the target specified by the element. A user's phrase matches the text of a **<menu>** or **<choice>** element if it comprises an attention phrase followed by one or more words from the text in the same sequence that they appear in the text (but not necessarily contiguously). An attention phrase is a phrase such as "go to" that indicates that a command follows.

Body Elements: A **<body>** element specifies some text to be converted to spoken output when the entity is visited:

<body name="name" next="target"> *text* **</body>**:

When a **<body>** element is visited, its text is spoken, and then the target specified by the next parameter is visited. The CML **<body>** entity with JSML markup serves a function for speech which is analogous to the function that an HTML body entity with presentation-oriented markup (such as headings, lists, etc.) provides for GUI. For information-retrieval applications (such as e-mail and news), a **<body>** element will typically contain information that the user is seeking. A body element is preferably marked up using Java Synthesis Markup Language (JSML) so as to obtain accurate and efficient text-to-speech synthesis:

<body>*Text to be spoken, marked up using* ***<EMP>****JSML****</EMP>*****</body>**

The "next" attribute of the **<body>** element has a value that is a target address (e.g., URL) which specifies another CML element (such as a **<menu>** element or a **<form>** element)

that is to be visited after the body has been spoken to the user. The following example will illustrate the function and operation of the **<body>** element and the use of JSML markup and the "next" attribute.

```
<body next="#menu1">
    <JSML>
    Welcome to the IBM ViaVoice<EMP>Speech Browser</EMP>. This tutorial will
        familiarize you with the spoken commands that you can use to control your
        speech browser.
    </JSML>
<body>
```

The body is formatted using JSML to place emphasis on the term "Speech Browser." The "next" attribute in this case is a relative URL (or target address) that instructs the browser to visit the element with the name "menu1" on the current page.

As described above, in one embodiment of CML, the **<body>** and **<menu>** elements provide spoken presentation and navigation of static information using spoken menus. As with HTML, this level of functionality is sufficient for static information that can be organized as a set of interlinked CML pages. A much richer set of applications is possible, however, if the user can supply to the application information taken from large sets, such as search words, dollar amounts, dates, stock names, etc. Such sets of possible inputs are too large to be presented in a menu, so another mechanism for collecting the input at the client is required. Furthermore, a mechanism may be used to compute the application response "on-the-fly" at the presentation server rather than being stored as with menu and body elements. This can be done, for example, via CGI (Common Gate Interface) programs and servlet programs running on the server, or any other backend logic. In cases where a complex logic is hidden behind the transaction or the server application, such server side on-the-fly modification of the response is the only way to proceed (except of course when information about the logic is transmitted using the conversational skeleton methods described in the above incorporated IBM Docket No. Y0999-114 or if the information is transmitted t the client: structure skeleton to disambiguate parts of the dialog).

Collecting such user input may also be accomplished in CML using a **<form>** elements.

Form Elements: A **<form>** element collects one or more pieces of information, or fields, from the user. A **<form>** element is typically used for collecting information, such as names, addresses, telephone numbers, and any other types of information that would be impractical to present as a list of choices in a menu, and has the general form:

<form action="URL">
introductory message text
<field name="*name 1*" **rule**="*JSGF*">*prompt text 1*</**field**>
<field name="*name 2*" **rule**="*JSGF*">*prompt text 2*</**field**>
...
</form>

An action is a target URL used in a **<form>**. The values of the **<field>**s of the form are appended to the action as attribute-value pairs, in the same manner as in HTML. A form groups together a set of user inputs that together are sufficient to warrant going back to a presentation server (which may in turn send a request to an information server) to obtain an application response. When a **<form>** element is visited, its title text (introductory text) is spoken, and the user is then prompted one by one for the values of any contained **<field>** elements. A grammar for matching the title text of the **<form>** element is activated when the **<form>** is loaded, and remains active thereafter. The <field> tag has the general form:

<field name="name" rule="ruleName" value="value">*prompt text*</**field**>

If a value has not already been supplied (either by the user or by the value parameter of the <field> tag) and if the prompt text is not empty, the user is prompted for the value of a field by speaking the specified prompt text. Acceptable responses for filling a field will depend on the input technology being used. For the "what you hear is what you can say" approach, acceptable responses includes a subsequence of the words of the prompt text followed by a phrase matching a specified grammar (which is preferably in Java Speech Grammar Format (JSGF)) that provides the set of possible values for the field. The portion of the response matching the grammar becomes the value of the field. As discussed below, this can be extended to NLU input. If the user's response fills a field, the user is then prompted for the next unfilled field, and so on.

When all the field values have been specified, the action target is visited using, for example, an HTTP GET method (i.e., the results are transmitted to the presentation server), an a page of CML is returned containing an entity which is then visited. It is to be appreciated that the user may fill in the value of any field either when prompted or at any point as long as the form is still in scope by speaking a portion of the prompt followed by a legal value of the field (i.e., by speaking the prompt text followed by a phrase matching the ruleName specified by the rule parameter, wherein the "ruleName" is a fully qualified JSGF rule name).

The following example will illustrate the function and operation of a form element.

<form action="*HTTP://localhost:8080/servlet/McServlet/*">

Please complete the order form.

<field name="sandwich"**rule**="*mcgrammar.sandwich*">Sandwich is?</**field**>

<field name="drink"**rule**="*mcgrammar.drink*">Drink is?</**field**>

<form>

When the above form is visited, the conversational browser outputs the introduction: "Please complete the order form." The user is then prompted "Sandwich is?". The user may then respond "Sandwich is chicken sandwich" or "Sandwich is hamburger" based on the following: The specified JSGF rule, "*<mcgrammar.sandwich>*", is found in JSGF grammar "mcgrammar", which reads

grammar mcgrammar;

public <sandwich> = hamburger chicken sandwich;

public <drink> = coke pepsi;

Note that the user may also fill in this or any field by saying "sandwich is sandwich" or "drink is drink" before being prompted for the value, and for as long as the sandwich form remains active. In this case, the browser selects a value for the field and then prompts the user one by one for the unfilled fields. This is a dialog feature known as "mixed initiative." This means that the user may take the initiative by saying "sandwich is sandwich," or the system may take the initiative by prompting the user for sandwich because, for example, the user has activated a menu choice that leads to a food-ordering form.

CML forms are similar to HTML forms, and in one embodiment, CML uses a mechanism similar to HTML to transmit the field values to the server: when the user completes the form, specifying a value for each field, the browser appends the *name=value* pairs for each

field to the specified form action URL, and the resulting URL is requested from the server via HTTP. Preferably, the form action URL incorporates the name of an application-specific function, such as a CGI script or a Java Servlet, which the server will call on to process the *name=value* pairs and return a response.

As with the case of a graphical web browser wherein the response to a completed form will be a page of HTML to be presented to the user on a display, with the conversational browser, the response will be a page of CML to be presented to the user using speech (by visiting the specified body, menu, or form entity). In either case, the new page will affect the interpretation of subsequent input actions.

Dialog Issues: One consideration in the design of a spoken dialog application is the ability to allow a user to readily determine what he/she can say at any point in the "conversation." Ideally, the application would accept any reasonable spoken input from the user (NLU), which is sometimes difficult to implement. Instead, an application can be designed to accept a limited set of inputs. This design, however, presents the user with the burdensome task of having to learn and remember a "language" comprising an arbitrary subset of his/her natural spoken language.

The CML browser advantageously mitigates this problem by implementing various approaches. One approach is referred to as the "what you hear is what you can say" approach, in which (as explained above) acceptable spoken inputs always echo the spoken prompts that are presented to the user. Each time a CML element is visited, the set of user inputs that the browser will accept is changed. For example, after the user visits a **<menu>** or **<form>**, the browser will accept responses appropriate to that menu or form. Following the "what you hear is what you can say " approach, the inputs accepted are in general echoes of the menu or form prompts, or some abbreviated version of the prompt.

For example, if the user hears: "Choose from the main menu: E-mail. Stock Quotes," the user can say: "Go to [the] main menu"; "Go to stock quotes"; and/or "Go to e-mail." In addition, if the user hears "You have 2 messages: New message 1 from Bill; Smith about Golf tee time. Message 2 from Jane Jones about Project meeting", the user can say: "Go to new messages"; "Go to new message 1"; "Go to message 2"; "Go to message from Bill"; "Go to message about tee time"; and/or "Go to message about project." Moreover, if the user hears: " Stock quotations.

Stock symbol is?", the user can say: "Go to stock quotations"; "Go to quotations"; "Stock symbol is I B M"; "Symbol is I B M" and/or "Stock is I B M."

As described above, the first two examples above may be accomplished in CML using a **<menu>** element that contains some title text ("Choose from the main menu") and a number of **<choice>** elements containing some prompt text ("E-mail", "Stock quotes"). The last example may be accomplished using a **<form>** element that contains some title text ("Stock quotations") and one **<field>** element containing some prompt text ("Stock symbol is?"). Accordingly, the title text of the **<menu>** or **<form>** and the prompt text of the **<choice>** or **<field>** define what the user may say.

Other options for implementing the dialog may be used. For instance, it is possible to pre-load answers to some of the forms. Namely, some scripting elements can be introduced to pre-program via procedures or script based on the answers or the selection from the user (i.e. variables are, for example, the fields of the prompts). Of course, this scripting can use/refer to a variable assigned by other event and procedures local or distributed (e.g. date, result from a query to the presentation server, etc...). It implies catching events, adding logical operations, loops and redirection (e.g. go to) statements. For situations using multiple forms, the capability to catch events is needed. Such events in the context of CVM could catch events also coming from applets and other conversational procedures, or scripts. Scripting can be done in a manner similar to Javascript or ECMAscript. The script may be embedded within the form **<script>...</script>.** This means also that forms, menus, etc., can throw and catch events. This enables appropriate programming capabilities. Scripts can be embedded anywhere within a CML page. Procedures, dialog components, conversational foundation classes and other services can be implemented by **<object>...</object>** tags.

In another aspect, a "Say what you heard" approach allows the user to be in control of the conversation (rather than the browser as described above). More specifically, during a conversational session, every previously visited **<menu>** or **<form>** that has been heard remains active. By way of example, assume the user has heard the main menu choices for "e-mail" and "news." With the "Say what you heard approach," the user can either immediately select the e-mail application by saying "select e-mail" or, at any time subsequent to hearing the main menu, select the news application by saying "go to news" without having to navigate back to the main menu. If the user has forgotten selections that were on the main menu, the use can get back to

the main menu by saying "go to main menu". This principle applies to every menu and form that a user has heard.

Another approach is "Say what you will hear," wherein the menus and forms become active before being visited and remain active throughout the browser session. More specifically, a menu or form becomes active when it is first loaded by the browser, which may be before the menu or form has been visited such as in the case where a page containing several menus and forms is loaded. Even if only one of the menus or forms on that page is visited (depending on the specifics of the URL), generally, all the menus or forms of the page will become active. This is illustrated in the following example:

```
<page>
  <body text="#new2menu">
    New message 2 from Steve about Pay raise.
    Bruce,
    We've decided to give you a 250% pay increase this year.
    Don't spend it all in one place.
    Steve
  </body>

  <menu name="new2menu">
    Choose from the message menu
    <choice target="#new3">Next message.</choice>
    <choice target="#new2forward">Forward message.</choice>
  </menu>
</page>
```

In the above example, assume the user requested to hear message 2. The conversational browser will visit the URL for the page (i.e., it will fetch the page and visit the first unit on the page), which in this case is the body of the message. After the message is finished, the "next" attribute of the **<body>** element causes the conversational browser to visit the "new2menu", which offers the user a couple of choices for disposition of the message. The user does not have to wait, however, to hear the menu before selecting one of the choices. Instead, the user may interrupt the message at any time to say, for example, "go to next message," since the menu

becomes active when the page is first loaded. Advantageously, this feature is useful in frequently used applications where the user learns to anticipate menus and forms that have not yet been presented. It is to be appreciated that the "say what you will hear" approach may be implemented using the skeletons as described in the above-incorporated IBM Docket No. Y099-114 to preload a dialog object when the conversational browser is built above CVM using Conversational Foundation Classes. Alternatively, the dialog object or dialog component can be implemented in CML.

It is be be appreciated that the "say what you heard" and "say what you will hear" mechanisms contribute to a dialog feature known as "mixed initiative." This means that in some cases the computer takes the initiative in the conversation - for example when the browser prompts the user for a set of menu items - and sometimes the user take the initiative - for example, when the user ignores the prompts and selects a menu item that was heard in the past, or interrupts the computer and selects a menu item that hasn't been presented yet. Thus, the set of spoken responses that are acceptable by the browser is defined by the entire set of CML elements visited and/or loaded by the browser, and not just by the most recent element visited. However, this can sometimes lead to unexpected results because of menus or forms that are no longer relevant being active. Consequently, CML according to the present invention offers a mechanism to limit the scope of activation for menus and forms.

Scope: Scope is a feature that allows a CML application designer to control the duration for which the spoken input associated with a particular **<menu>** or **<form>** is active. The "scope" feature is specified by a scope attribute that may be associated with a **<menu>** or **<form>** element:

<menu name="name" scope="scope"> *title text, choices* **</menu>**

<form name="name" action="target" scope="scope"> *title text, fields* **</form>**

Exemplary values of the "scope" attribute are the following:

Local: The menu or form is only active if the menu or form itself is the last element that the browser visited.

Page: The menu or form is only active if the last element visited by the browser is on the page (or a subpage of) the page that directly contains the menu or form.

Global: This is a default value, wherein the menu or form is active for the entire duration of the browser session beginning when it is first loaded by the browser (even if the form or menu itself has not yet been visited).

Application specific: This refers to all the page identified with the site or the application (e.g., via meta-information tag).

In many cases, the greatest flexibility for the user is provided if a menu or form becomes active as soon as the browser has first seen the menu or form and remains active thereafter. For example, as explained above, it is advantageous to have main menus or major topic menus active for the entire sessions so that a user can jump directly to the choices provided by the menus without first having to navigate back through a "maze" of menus. In some circumstances, however, this can decrease speech recognition accuracy and produce unexpected results. Consider the following example.

```
<page name="new2">
  <body next="#new2menu">
      New message 2 from Steve about Pay raise.
      Bruce,
      We've decided to give you a 250% pay increase this year.
      Don't spend it all in one place.
      Steve
  </body>

  <menu name="new2menu" scope="page">
      Choose from the message menu.
      <choice target="#new3">Next message.</choice>
      <choice target="#new2forward">Forward message.</choice>
  </menu>
```

The menu associated with each message allows the user to say "go to next message." This selection, however, probably makes little sense after the user has left the e-mail message to do another task, and it could be very surprising if it were activated as the result of a speech recognition error. Therefore, in this example, the message menu is given a scope of "page," meaning that it will only be active as long as the last element visited by the browser was in the

page that contains the message menu. In this case it means that "go to next message" is only active as long as the last browser action involved visiting the body of the e-mail item.

Additional use of the scope and behavior with the scope are provided in the embedded CML (as discussed below) Another scope tag that may be used in conjunction with CML is "Multiple". A discussion on the use of multiple form (not in CML form) is provided in: Kishore, et al, "Free-Flow Dialog Management Using Forms," Proc. Eurospeech 1999, Budapest Hungary, September 1999 and Davies et. al., "The IBM Conversational Telephony System For Financial Applications, Proc. Eurospeech 99, Budapest Hungary, September 1999. Note that according to this, instead of having multiple pages active, multiple forms can be simultaneously activated on a page. These pages can be explicitly nested in the document or they can be referenced by address (to be loaded). Similarly some of these forms can be presented as procedural components.

Typically as described above, under one NL modality, the NL dialog is implemented by activating multiple forms. The forms are filled simultaneously until all the mandatory fields of one form are filled. When filled, the corresponding action or query is executed. When multiple forms are filled, disambiguation dialog is added (e.g. by activating a new form). Multiple scope forms are typically loaded via a page which indicates to load these different forms:

<menu name =mainnl scope =global>

Introduction dialog

<form name = form1 scope = multiple>....</form>

<form name = form2 scope = multiple> ...</form>

<form name = form3 scope = multiple> ...</form>

...

</menu>

Upon any input that implies an action, a new page will be loaded after completion of the action:

<menu name=mainnl scope = global> (we "update" the global menu)

<form name = form1 scope = multiple context = maintain> ...</form>

<form name = form2 scope = multiple context = reset>....</form>

<form name = form3 scope = deactivated context = reset> </form>

<form name = formnew scope = multiple context = reset>....</form>

...

</menu>

Alternatively all these forms can be loaded by :

<form name ... scope ... load = target> </form>

When a load argument is present, the content of the form is to be downloaded from the target and placed between the **<form> </form>** tags. The same concept can be used for any other element of the language. Typically, the menu is updated and new forms are added and some of the forms can be deactivated (scope deactivated).

The "context" tag in the above example indicates how to handle past conversation history. In the two examples given above, the form with a context "maintain" will keep the value filled in the form from the previous steps of the dialog (previous input from the user). The "reset" tag refers to resetting the values to default values. Note also that scope can be defined with a duration stamp: scope = 5 s or scope = 1 minute or scope = the name of a scope. The scope is introduced as

<scope name=scope1 status = active> </scope>

When not defined, a scope is non-active. When defined, it can be active or non-active. Elements scoped with that name are activated when the scope is activated and deactivated otherwise.

It is to be understood that the forms used here can be implemented with a procedural object:

<form name=form3 scope = multiple context = reset load = target> </form>

where the target page contains applets, procedures, services etc for example:

<Dialog Object>

<applet ...>

</applet>

</Dialog object>

Again, a dialog object can be an applet (java), a script (interpreted by the browser), a conversational object (Conversational dialog written on top of CVM using conversational foundation classes), plug-in or servlet activation, or any other procedural implementation or service providers.

In the present case, multiple objects can be loaded in parallel. This is typically a case where the browser is implemented on top of CVM, w here the CVM is able to register these

different objects and their context and to determine which dialog is active as a function of the user input.

This last item illustrates that with CML and a browser hooked to appropriate dialog management capabilities, CML can be used to design NL dialogs, independently of the underlying NL technology. Form based, procedural based (decision networks) and grammar based NL dialog managers are fully supported by this approach.

It is also possible to have overloaded tags (i.e., modifying the action associated with a tag). Typically, this can be done via XML or other mechanisms allowing a description of the new meaning of the tag and the scope of the overloading definition.

Browser features: So far we have discussed CML and described how a browser behaves, with respect to spoken output and spoken input, in response to the markup language. The following discussion is directed to features of the conversational browser that are not intrinsic to CML, but rather are characteristic of the browser implementation.

To begin, the conversational browser preferably implements a number of built-in commands (which are similar in spirit to the built-in functions presented on menus and toolbars in a visual browser). Examples of such commands are as follows:

[be] quiet; shut up: These commands will cause the Browser to stop a current spoken output and wait for further instructions from user. This feature is known as "barge-in" and applies to all spoken input so that a user may interrupt the browser at any time. So, for example, the user may interrupt the browser by saying "quiet please," in which case the browser will stop the current output and wait for further instruction. To cause the browser to repeat the interrupted output, the user may then utter, for example, "repeat that please." The user may also interrupt the browser at any point with a spoken command, such as a menu item selection.

say again; repeat that: These commands will cause the Browser to repeat a most recently visited element (menu, form, or body).

go back: This command will cause the Browser to return to a previously visited element in history list.

go forward: This command will cause the Browser to advance to a next visited element in history list. (Only meaningful after some number of "go back" commands).

[[[go]to]the]beginning: These commands will cause the Browser to go to the first element visited (e.g., the "home" page).

The brackets in the above exemplary commands indicate optional words. Any of these commands may be preceded or followed by "please." Other classical commands can be defined for the browser or added by the user.

Inevitably, ambiguities may arise between these various types of spoken input. Ambiguity resolution is handled at the level of units (forms and menus) by maintaining a most-recently-used (MRU) queue of units. More specifically, a spoken input is resolved against a unit if it matches a spoken phrase allowed by the unit (attention phrase followed by title text or choice prompt, field prompt followed by legal field value). The browser attempts to resolve every spoken input against each unit in the MRU queue in order. In one embodiment, the MRU queue may be maintained as follows:

1. When a spoken input is resolved against a unit in the queue, that unit is moved to the head of the queue.

2. Then, if the spoken input causes a target to be visited, all the units in the page containing the target are moved to the head of the queue.

3. Finally, the unit corresponding to the visited target is moved to the head of the queue.

It is to be understood that with a browser built on top of a CVM, the CVM will perform the appropriate ambiguity resolution and dialog with the user if any ambiguity is suspected to arise as discussed in the IBM Docket No. YO999-111P. Essentially, once a CML specification is produced, it will virtually prescribe the manner in which a dialog should be handled. If this needs to be overwritten, or if other behavior(s) need to be introduced, this can be performed by express calls to the underlying CVM, OS, or platform (if available).,

Plugins: The present invention provides a mechanism that provides a plugin- or applet- like capability. A URL specifying a **<choice>** target or a **<form>** action may invoke local plugin code by using the local: protocol, following one of several exemplary forms:

local:service/function; or

local:service/function?arg1=value1,arg2=value2,...

The first form is useful for **<choice>** targets which have no arguments and for **<form>** actions where the arguments are supplied by the browser from the values of the contained

<**field**>s. The second form is useful for <**choice**> targets where the arguments are supplied in the URL.

In a preferred embodiment of the browser implementation using Java, a local: URL is implemented by looking up the service in a local table that maps it to a class name, instantiating an object of that class if it is not already instantiated, and then interpreting the function as the name of a method to be called. The table of services is preferably located in the file services of the conversational browser, wherein the file itself contains a description of its format. This includes registration of all the available services and other objects (procedural or not)

The plugin code invoked by the local: URL returns a string containing the CML document represented by the URL. Visiting a local: URL causes the returned document to be interpreted in just the same way as visiting any other URL. It is to be appreciated that these features may also be implemented using applets (downloaded or locals), conversational objects and servlets/cgi as well as distributed applications (with for example socket connections and RPC protocols) to act as a local application.

The service may subsequently interact asynchronously with the browser (e.g., to notify the user asynchronously of important events) by starting a thread and causing the browser to asynchronously visit specified URLs (including local: URLs) via an API provided by the browser. The browser API allows local plugin code to cause the browser to visit a specified URL, just as if the user had spoken something that caused the browser to visit the URL. This can be used to provide asynchronous user notifications based on local events. The local plugin will extend the class CML, Service, from which the local plugin inherits a method visit which, when invoked, causes the browser to visit the specified URL with the specified parameters: public class Service {public void visit(URL url, Dictionary parameters) throws IOException;}.

Considerations for Embedded Conversational Browser Applications

Registration mechanism: In situations wherein the functions of an embedded conversational browser may be limited due to inadequate resources (memory, CPU power, etc), a cookie mechanism can be used that allows the browser to transfer to a server a description of its capabilities. For example: Speech_Reco = 1:Speaker_Reco = 0; TTS = 1; Dialog_Manager = 0,NLU = 0; Speech_Reco.Vocabulary_Size=500; Speech_Reco.FSG=1 etc. Upon receiving the cookie, the server side can examine its content and then modify the CML page accordingly. For

example, if the cookie indicates that the vocabulary size is 50, the CML menus can be generated with very limited vocabularies (e.g. selection of a number per item instead of the actual link).

In another embodiment, rather than using a cookie, registration conversational protocols can be used as described in the above incorporated patent applications IBM Docket Nos. YO999-111P and YO999-113P. In such an embodiment, objects are exchanged which describe the characteristics and properties of the engines as well as possibly the requirements of the application. The handshake will determine the conversational responsibilities of the applications. Again, in such case, the preliminary handshake can use HTTP or other protocols such as RPC, RMI, etc. This is especially important for Dialog Objects that can directly dialog with the server to check an possibly adapt their behavior to the capabilities of the local browser. Note also that like in a conventional browsers, these cookies (or procedural protocols) can perform other tasks such as customization of the presentation or service to the user's preference or persistence of these preferences or of the history across sessions. In other words, if it is an option selected by the user, persistence across devices of the context history is guaranteed by having cookies with the past history of a site uploaded upon connection. When performed via a CVM or procedural objects, the information is transmitted via procedural protocols. Now ID cookies are also used to guarantee persistence across devices when stored on the server. When connecting from different devices, the cookies (or procedural protocol) can be stored on the server or the presentation server. Identification of the user can be done using conventional techniques such as userID, caller ID, speaker recognition or speech biometrics. Any other meta-information (persistence, user preference, application preference, and usage history, context, etc) must be transferred (via a cookie or via conversational protocols)

Therefore, if the conversational engine cannot generate the baseforms, the tag can provide a baseform. Various approaches can be utilized: 1) upon determination of the browser capabilities, the browser is sent a different CML page; 2) a transcoder can dynamically modify a CML page sent to the browser based on the registered capabilities; or 3) a CML page can be modified to send the speech and perform complex functions on a networked server (as discussed in IBM Docket No. YO999-113P).

Dynamic deactivation: The conversational browser may use dynamic activities of CML. This is another solution which allows modification of the behavior of the conversational browser for

embedded applications. In particular, instead of completely following the paradigm of "what you hear is what you can say," the conversational browser may only activate locally the markup unless explicitly limited by CML. Furthermore, upon reaching the maximum amount of supported active vocabulary, the speech browser can progressively and hierarchically be deactivated: the oldest command which is not a browser shell command or a global command (as defined by the CML with a Global Tag **<Global></Global>** or priority level tags; this is equivalent to a concept of scope tags). A top menu remains active as long as possible commands under a menu item are deactivated first etc. Deactivated commands are cached in series of command caches. When the likelihood (or other confidence measure of the recognized input) obtained for a command are too low or the command is rejected by the user, the utterance is re-decoded against the most recent command cache etc until an acceptable recognition is obtained. Recently, efforts have been initiated to develop appropriate confidence measures for the recognized speech. For instance, In "LVCSR Hub5 Workshop," April 29 - May 1, 1996, MITAGS, MD, organized by NIST and DARPA, different approaches are proposed to attach a confidence level to each word. One method uses decision trees trained on word-dependent features (amount of training utterances, minimum and average triphone occurrences, occurrence in language model training, number of phonemes/lefemes, duration, acoustic score (fast match and detailed match), speech non-speech), sentence-dependent features (signal-to-noise ratio, estimates of speaking rates: number of words or of lefemes or of vowels per seconds, sentence likelihood provided by the language model, trigram occurrence in the language model), word in a context features (trigram occurrence in language model) as well as speaker profile features (accent, dialect, gender, age, speaking rate, identity, audio quality, SNR, etc.). A probability of error is computed on the training data for each of the leaves of the tree. Algorithms for building such trees are described in Breiman et al, "Classification and regression trees," Chapman & Hal,1993. At recognition, all or some of these features are measured during recognition and for each word the decision tree is walked to a leave which provides a confidence level. In the reference by C. Neti, S. Roukos and E. Eide entitled "Word based confidence measures as a guide for stack search in speech recognition," ICASSP97, Munich, Germany, April, 1997, a method is described which relies entirely on scores returned by an IBM stack decoder (using log-likelihood -actually the average incremental log-likelihood, detailed match, fast match). In the LVCSR proceeding, another method to estimate the confidence level is performed using

predictors via linear regression. The predictor used are: the word duration, the language model score, the average acoustic score (best score) per frame and the fraction of the NBEST list with the same word as top choice. The present embodiment offers a combination of these two approaches (confidence level measured via decision trees and via linear predictors) to systematically extract the confidence level in any translation process, not limited to speech recognition.

A "rewind" or "reload" command can be implemented to read back to the user all the commands met on a page. It follows the same principal of dynamic deactivation. More specifically, a entire voice navigation interface is provided by the browser to enable navigation through the previous forms and menus. It can be implemented by a command and control interface such as IBM's ViaVoice VoiceCenter (which performs Command and Control for Windows).

Capability-based frames: Similarly to the frame structure, the CML developers can offer multiple dialog based on different levels of capabilities [cfr<frame> and </frame> tags in HTML].

Forms with minimum requirement: Of course only forms (or menus or other components) which have a limited set of possible entries (limited vocabulary and FSG, selected menu) can be used with the embedded engines. To maintain the capability to have forms, the browser offers at least two other modalities: i) form filling with prescribed grammar (compilation done on the server) and vocabulary list: the other commands, except the <Global> commands are deactivated, ii) applet capturing the audio, computing special features and sending them to a server indicated by a URL, iii) applet to send or receive raw audio to the server. The selection of the form filling method is done by the server or by the browser. This requires that each grammar/vocabulary list to load must contain also a description of the engine requirements. (ii and iii need server/communication capabilities as described in YO999-113P).

A tool will be needed to determine roughly the engine minimum requirements for a given FSG, for the transcoding approach as well as the multiple pages or frame approaches.

MULTI-MODAL BROWSER

With multi-model browsing, HTML and CML can provided on the same page or sent separately and synchronized. Tags to distinguish what is to show on a visual browser (<GUI>) vs. what to show on a multi-modal browser (<M-M>) and speech browser (<speech>): a GUI

browser displays everything while a multi-modal browser can display selectively some items. This is illustrated by the following examples:

<GUI>

.....

</GUI>

<Speech+GUI+M-M>

.....

</Speech+GUI+M-M>

<GUI+M-M>

<img = ...>

</GUI+M-M>

<Speech>

.....

<GUI>

.....

</GUI>

<M-M>

.....

</M-M>

<M-M+Speech>

.....

</M-M+Speech>

In other words, CML and HTML are preserved. Some dynamic HTML features and functions as well as combinations rules are added.

Additional tags to offer Dynamic ML (HTML and CML combined): For example as the TTS reads the text, the item change colors or a local background change color. Change of color of the link selected by voice etc... Selection of recognized text by voice (change color can be selected and modified).

In summary, the introduction of the conversational browser allows Internet and Telephone (IVR) to be unified and thereby decrease the cost, enlarge the coverage and flexibility of such applications. The architecture uses the same access protocols and the information structure (HTML, XML, CML and/or other ML such as WML). The present invention can be applied to many different business solutions. The main advantages are that it is easy to implement, high flexible, platform independent, uses the existing infrastructure, deployed and managed centrally, provides high security, low cost maintenance, and is readily expandable/scalable, all of which are typical for any Internet solutions. Advantageously, the application provider need support only one HW platform which provides ease of maintenance. The content provider can administer various applications from the same WWW server so as to service a large number of different clients (pure speech, pure GUI and a combination of speech/GUI), which provides ease of maintenance. Moreover, the Internet technology enables automatic updating of all servers on the network. The access rights can be also managed centrally from any place accessible over Internet. High level of security can be maintained. The current architecture can be extended to other clients in addition to regular telephones, such as GSM terminals. This is very attractive from the perspective of the investment that is required to maintain a service such as personal banking.

In addition, when only HTTP protocols are used, the traffic on the network between the browser (conversational/GUI) and the WWW server is minimized, the network is used for sending the completed replay in one batch. For example, while entering the account Number and the SSN only the browser/conversational browser are active, the network is idle. The browsers are intelligent enough to catch many errors before the server is queried. Error correction and local disambiguation can for example be performed via CVM services or downloaded applets. The information to the WWW server is sent in both cases using the same protocol the HTTP protocol. The replay information is sent back always using the same protocol and it is the task of the browser to get it to the user in the right form. The suggested architecture enables the designers of these applications to use applets, small programs (java or other conversational procedures, especially if the browser is written over a CVM)), which are executed in the browser to preprocess the input and output information. This further increases the flexibility of the application and furthermore decreases the load of the network. For the conversational browser we can easily imagine a custom "Text to Speech" applet will be sent over

and then the required bandwidth will be the same or smaller then the graphical one. In particular, this can be done to synthesize prompts in another language and recognize input in other languages.

Indeed, the present invention provides these advantages to the current telephony applications, typically the IVR applications. The telephone number is perfect for billing purposes. The latency of Internet does not pose any problem to IVR. In any event, latency can be reduced using services like Qos (quality of service) and RSVP (resource reservation protocol). The capability of connecting on the Internet to other servers (IVRs) makes the capabilities much larger. The server can be located anywhere on a computer network with the access to required speech recognition systems being also available anywhere on the network. The packet switching technology with the TCP/IP protocol utilized the net resources better then regular phone connections. The billing may be based on the quality of used channel. The switched packet network is cheaper and transfers less data, therefore, requiring less bandwidth. This will result in providing the services with lower cost. This concept gives anybody chance to set a server and put the information on net.

Although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present system and method is not limited to those precise embodiments, and that various other changes and modifications may be affected therein by one skilled in the art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A conversational browsing system, comprising:

a conversational browser comprising:

a command and control interface for converting an input command into a navigation request, the input command comprising a speech command; and

a CML (conversational markup language) processor for parsing and interpreting a CML file, the CML file comprising meta- information representing a conversational user interface for presentation to a user;

a conversational engine for decoding input commands for interpretation by the command and control interface and decoding meta-information provided by the CML processor for generating synthesized audio output; and

a communication stack for transmitting the navigation request to a content server and receiving a CML file from the content server based on the navigation request.

2. The conversational browsing system of claim 1, where the input command comprise multi-modal input.

3. The conversational browsing system of claim 1, further comprising a (CVM) conversational virtual machine, wherein the conversational browser executes on top of the CVM and makes requests for conversational services to the conversational engines through the CVM.

4. The conversational browsing system of claim 3, wherein the conversational browsing system executes in a client device, and wherein the communication stack comprises conversational protocols for registering conversational capabilities of the client device with the content server.

5. The conversational browsing system of claim 3, wherein one of the conversational browser, CVM and conversational engines are distributed over a network.

6. The conversational browsing system of claim 1, wherein the communication stack implements standard networking protocols for transmitting the navigation request and receiving the CML file.

7. The conversational browsing system of claim 6, wherein the standard networking protocols comprise one of TCP/IP (transmission control protocol/internet protocol), HTTP (hypertext transmission protocol), WAP (wireless application protocol), VoIP (voice over internet protocol) and a combination thereof.

8. The conversational browsing system of claim 1, further comprising a transcoder for converting legacy information formats of a content server to a CML file.

9. The conversational browsing system of claim 8, wherein the transcoder accesses and interprets logic information of an application to generate a dialog with a user through the conversational browser.

10. The conversational browsing system of claim 8, wherein the transcoder is executed on one of the conversational browser, a conversational proxy server, the content server, or distributed among a combination thereof.

11. The conversational browsing system of claim 8, wherein the transcoder generates a custom CML file based on conversational capabilities of the machine on which the conversational browsing system executes.

12. The conversational browsing system of claim 1, wherein the CML file comprises one of (1) a page elements that group other CML elements; (2) a body element which specifies output to be spoken by the conversational browser (3) a menu element comprising introductory text that is spoken to a user and a list of choices, each choice having prompt text and being associated with a target address identifying a CML element if the corresponding choice is selected; (4) a form element for inputting at least on item of information and a target address to sent the at least one item of information; and (5) a combination thereof.

13. The conversational browsing system of claim 12, wherein a target address comprises one of a URL (uniform resource locator) and a socket address.

14. A method for providing conversational browsing, comprising the steps of:

receiving an input command, the input command comprising a speech command;

decoding the input command with at least one of a plurality of conversational engines;

generating a navigation request based on the decoded input command for retrieving a CML (conversational markup language), the CML file comprising meta- information representing a conversational user interface for presentation to a user;

transmitting the navigation request and accessing the requested CML file using a standard networking protocol; and

parsing and interpreting meta information comprising the CML file to provide an audio presentation of the information content of the CML file.

15. The method of claim 14, further comprising the step of:

registering conversational capabilities with an entity from which the CML file is accessed; and

customizing the CML file based on the registered capabilities.

16. The method of claim 14, further comprising the step of:

generating a navigation request for accessing a file comprising a legacy information format; and

converting the legacy information format to a CML file.

17. The method of claim 16, wherein the step of converting is performed by a transcoding proxy associated with a content server from which the file is accessed.

18. The method of claim 14, further comprising the steps of accessing and interpreting logic information of an application associated with the CML file to generate a dialog.

19. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for providing conversational browsing, the method steps comprising:

receiving an input command, the input command comprising a speech command;

decoding the input command with at least one of a plurality of conversational engines;

generating a navigation request based on the decoded input command for retrieving a CML (conversational markup language), the CML file comprising meta- information representing a conversational user interface for presentation to a user;

transmitting the navigation request and accessing the requested CML file using a standard networking protocol; and

parsing and interpreting meta information comprising the CML file to provide an audio presentation of the information content of the CML file.

20. The program storage device of claim 19, further comprising instructions for performing the step of:

registering conversational capabilities with an entity from which the CML file is accessed such that the entity can customize the CML file based on the registered capabilities.

21. The program storage device of claim 19, further comprising instructions for performing the step of:

generating a navigation request for accessing a file comprising a legacy information format; and

converting the legacy information format to a CML file.

22. The program storage device of claim 19, further comprising instructions for performing the step of accessing and interpreting logic information of an application associated with the CML file to generate a dialog.

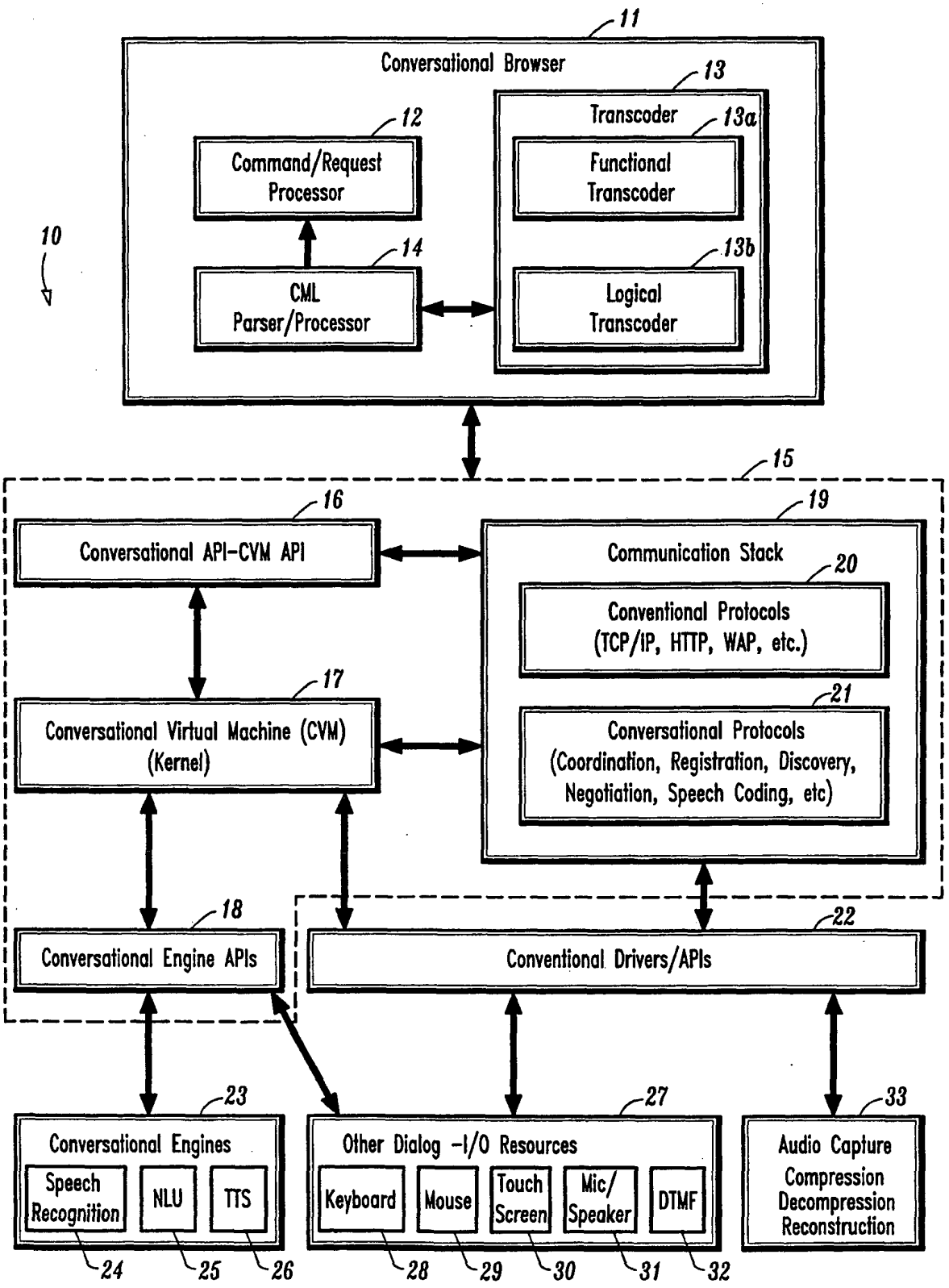

11
Conversational Browser
13
Transcoder
13a
Functional Transcoder
12
Command/Request Processor
10
14
CML Parser/Processor
13b
Logical Transcoder
15
16
Conversational API-CVM API
19
Communication Stack
20
Conventional Protocols (TCP/IP, HTTP, WAP, etc.)
17
Conversational Virtual Machine (CVM) (Kernel)
21
Conversational Protocols (Coordination, Registration, Discovery, Negotiation, Speech Coding, etc)
18
Conversational Engine APIs
22
Conventional Drivers/APIs
23
Conversational Engines
Speech Recognition
NLU
TTS
24
25
26
27
Other Dialog -I/O Resources
Keyboard
Mouse
Touch Screen
Mic/ Speaker
DTMF
28
29
30
31
32
33
Audio Capture
Compression
Decompression
Reconstruction

FIG. 1

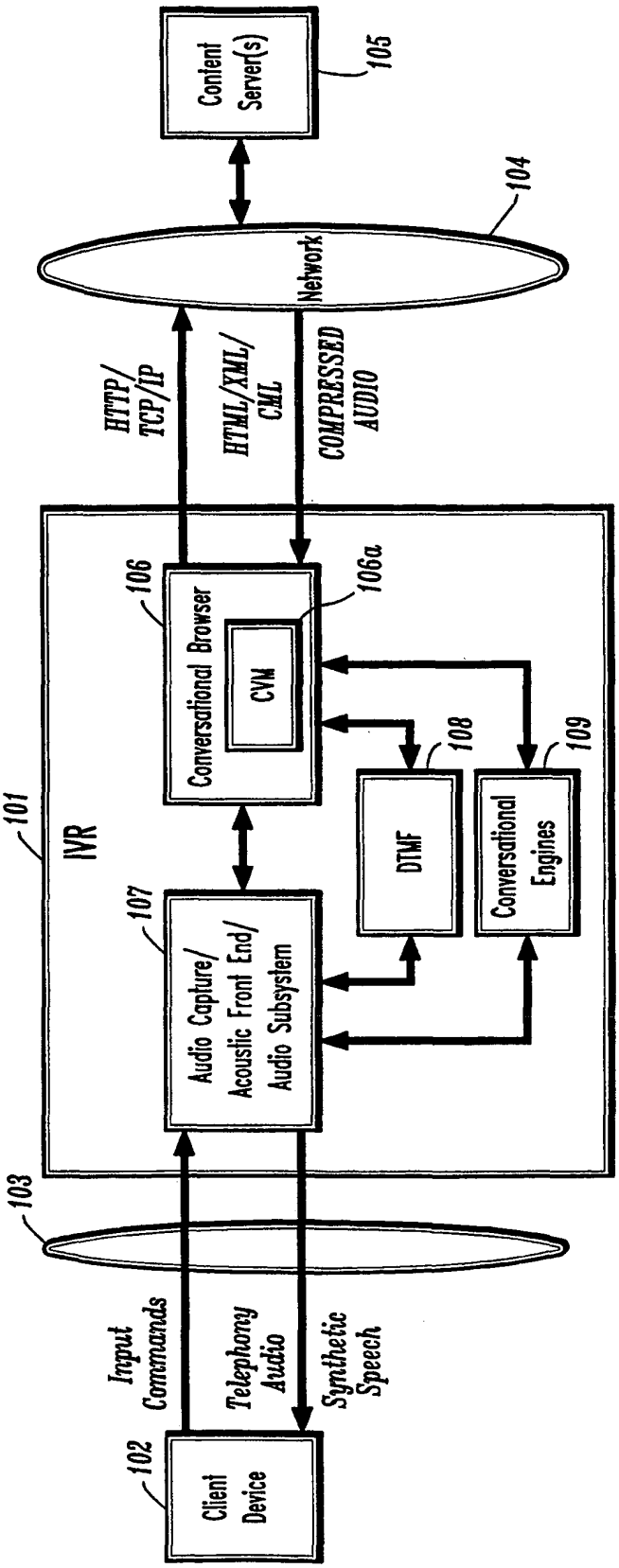

Content Server(s)
105
104
Network
HTTP/ TCP/IP
HTML/XML/ CML
COMPRESSED AUDIO
101
IVR
106
Conversational Browser
106a
CVM
108
DTMF
109
Conversational Engines
107
Audio Capture/ Acoustic Front End/ Audio Subsystem
103
Input Commands
Telephony Audio
Synthetic Speech
102
Client Device
100

FIG. 2

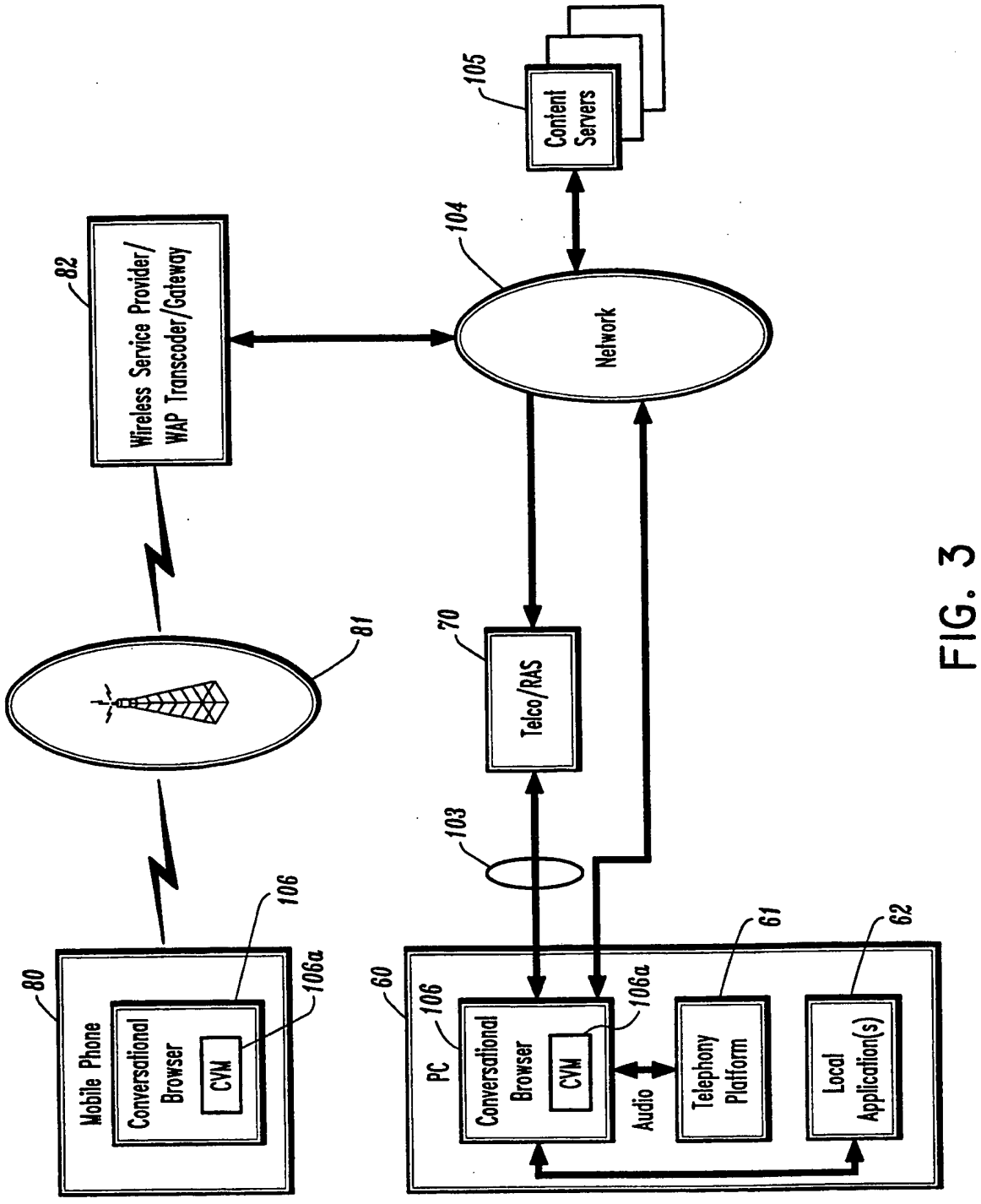

105
Content Servers
104
Network
82
Wireless Service Provider/ WAP Transcoder/Gateway
81
70
Telco/RAS
103
80
106
106a
Mobile Phone
Conversational Browser
CVM
60
PC
106
Conversational Browser
CVM
106a
Audio
61
Telephony Platform
62
Local Application(s)

FIG. 3

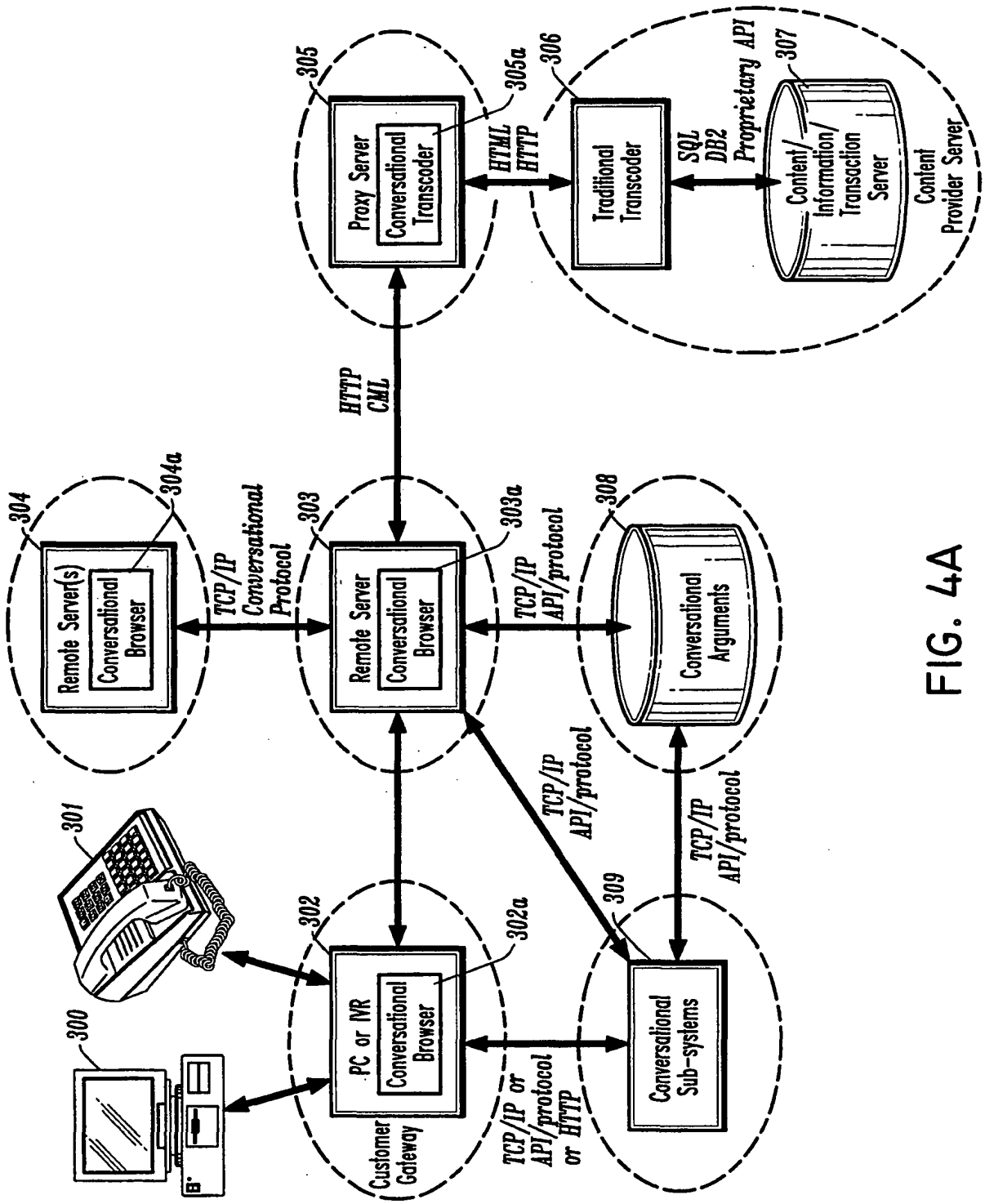

305
Proxy Server
Conversational Transcoder
305a
HTML HTTP
306
Traditional Transcoder
SQL DB2 Proprietary API
307
Content/ Information/ Transaction Server
Content Provider Server
HTTP CML
304
304a
Remote Server(s)
Conversational Browser
TCP/IP Conversational Protocol
303
Remote Server
Conversational Browser
303a
TCP/IP API/protocol
308
Conversational Arguments
TCP/IP API/protocol
TCP/IP API/protocol
301
302
PC or IVR
Conversational Browser
302a
309
Conversational Sub-systems
300
Customer Gateway
TCP/IP or API/protocol or HTTP

FIG. 4A

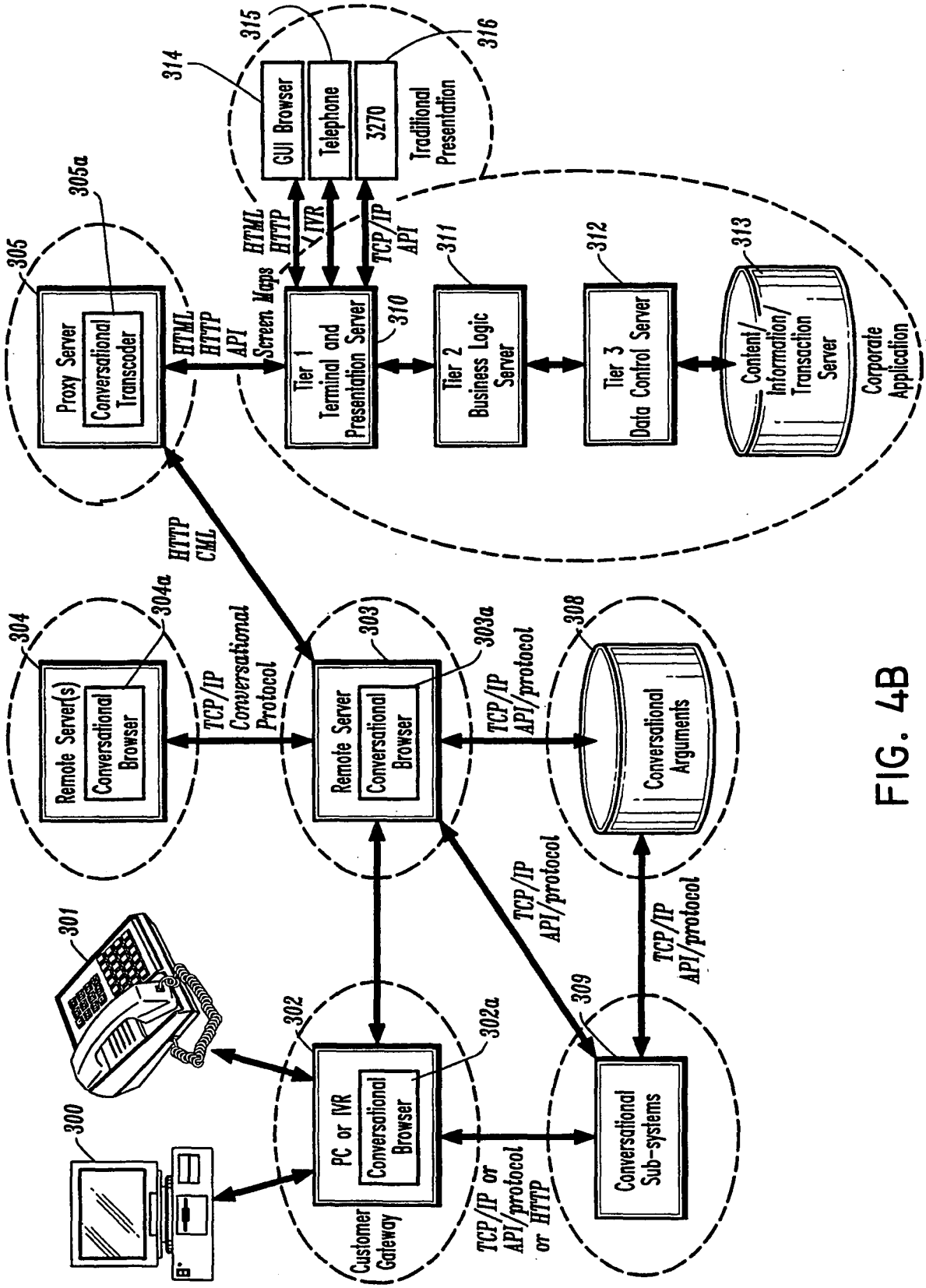

315
316
314
GUI Browser
Telephone
3270
Traditional Presentation
305a
305
HTML
HTTP
IVR
TCP/IP
API
Screen Maps
311
312
313
Proxy Server
Conversational Transcoder
HTML
HTTP
API
Tier 1
Terminal and
Presentation Server
310
Tier 2
Business Logic
Server
Tier 3
Data Control Server
Content/
Information/
Transaction
Server
Corporate
Application
HTTP
CML
304
304a
303
303a
308
Remote Server(s)
Conversational
Browser
TCP/IP
Conversational
Protocol
Remote Server
Conversational
Browser
TCP/IP
API/protocol
Conversational
Arguments
TCP/IP
API/protocol
TCP/IP
API/protocol
301
302
302a
309
PC or IVR
Conversational
Browser
Conversational
Sub-systems
300
Customer
Gateway
TCP/IP or
API/protocol
or HTTP

FIG. 4B

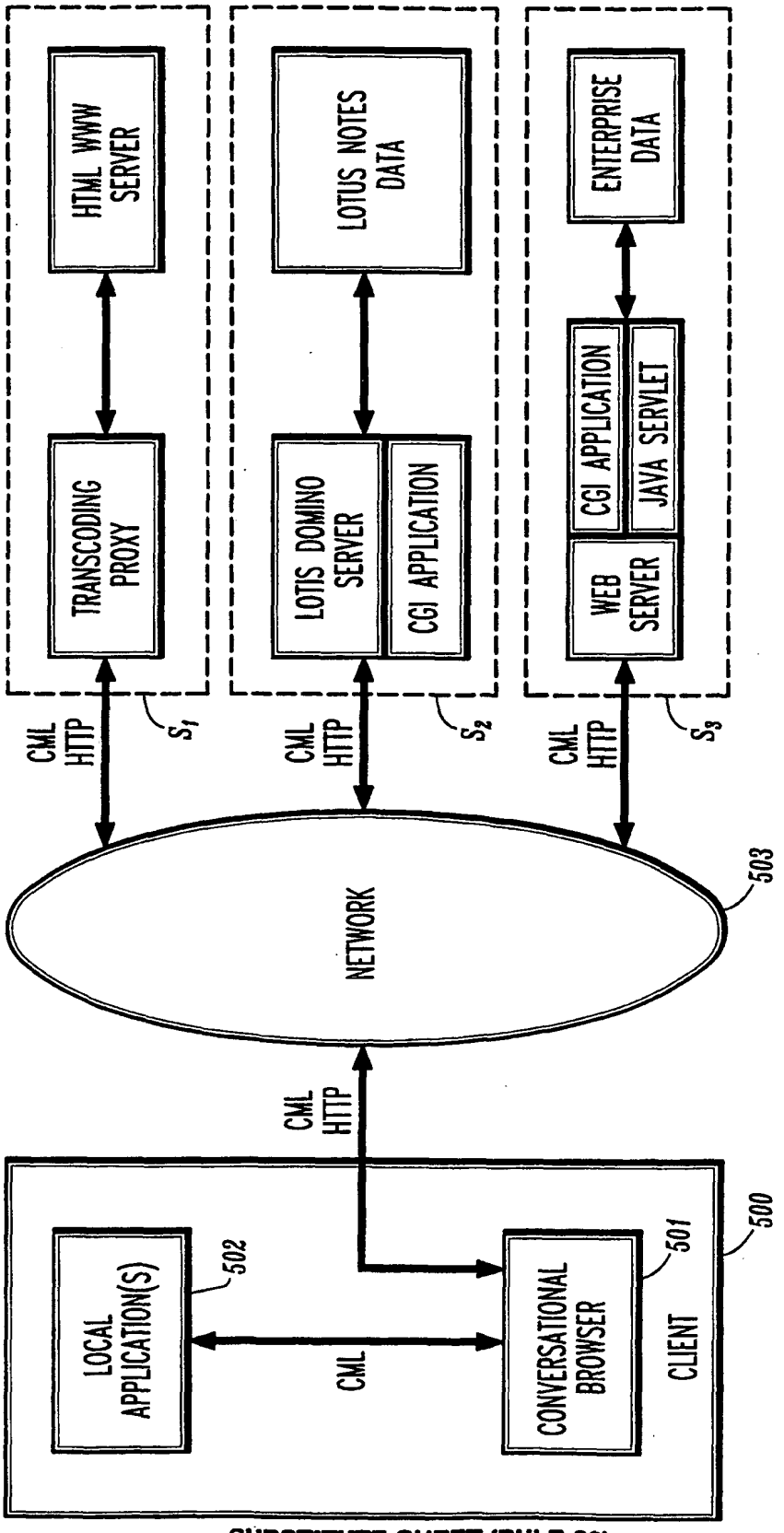

HTML WWW SERVER
TRANSCODING PROXY
LOTUS NOTES DATA
LOTIS DOMINO SERVER
CGI APPLICATION
ENTERPRISE DATA
CGI APPLICATION
JAVA SERVLET
WEB SERVER
CML HTTP
S_1
CML HTTP
S_2
CML HTTP
S_3
NETWORK
503
CML HTTP
LOCAL APPLICATION(S)
502
CML
CONVERSATIONAL BROWSER
501
CLIENT
500

FIG. 5

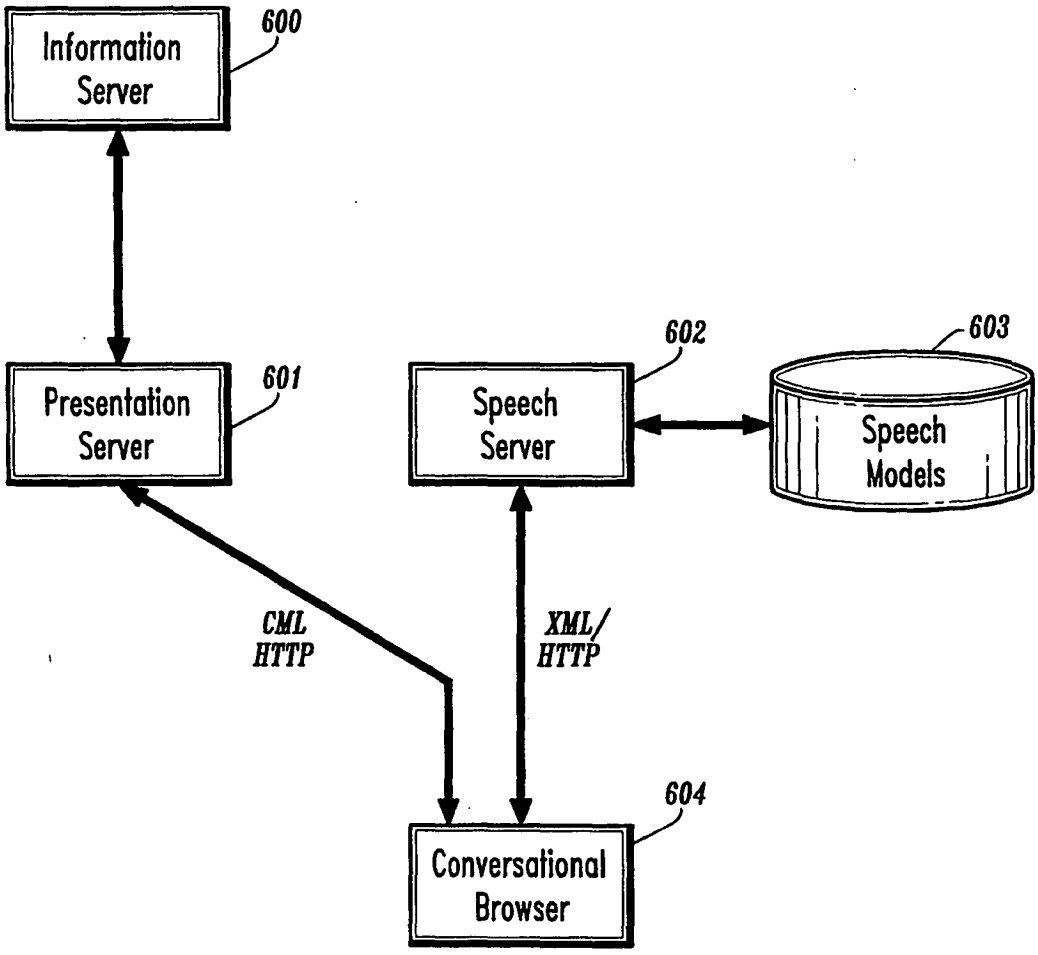

Information Server
600
Presentation Server
601
Speech Server
602
Speech Models
603
CML HTTP
XML/ HTTP
Conversational Browser
604

FIG. 6

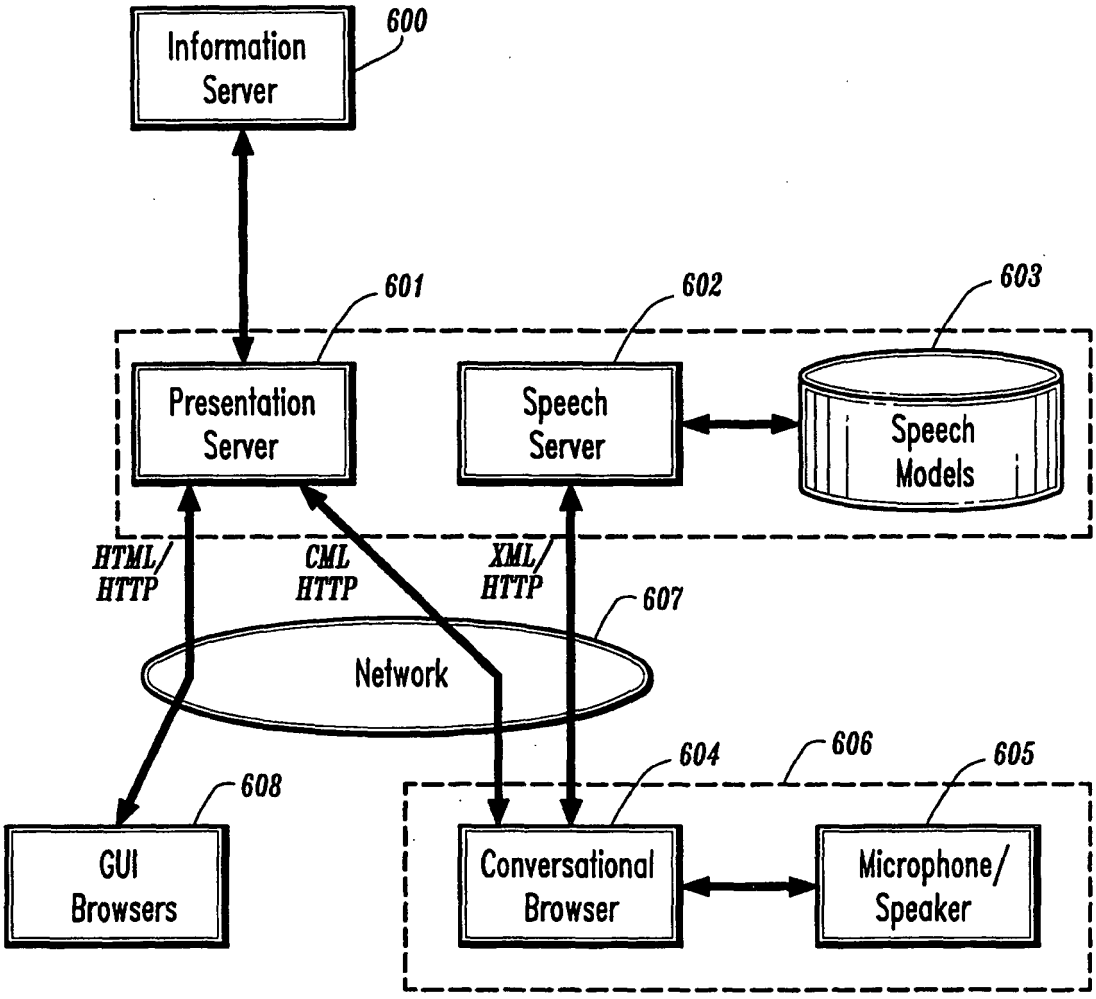

Information Server
600
601
602
603
Presentation Server
Speech Server
Speech Models
HTML/ HTTP
CML HTTP
XML/ HTTP
607
Network
604
606
605
608
GUI Browsers
Conversational Browser
Microphone/ Speaker

FIG. 7

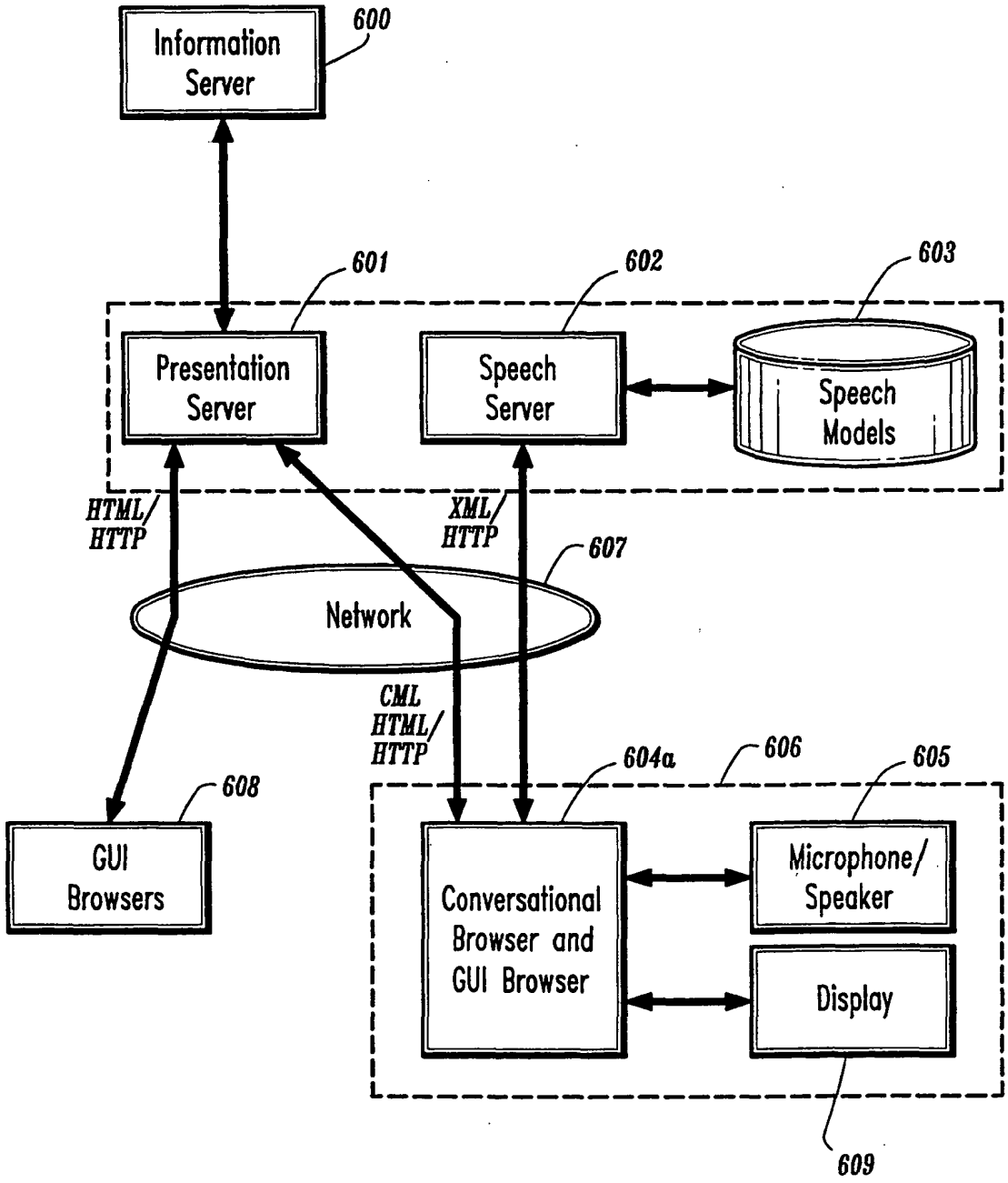

Information Server
600
601
602
603
Presentation Server
Speech Server
Speech Models
HTML/ HTTP
XML/ HTTP
607
Network
CML HTML/ HTTP
604a
606
605
608
GUI Browsers
Conversational Browser and GUI Browser
Microphone/ Speaker
Display
609

FIG. 8

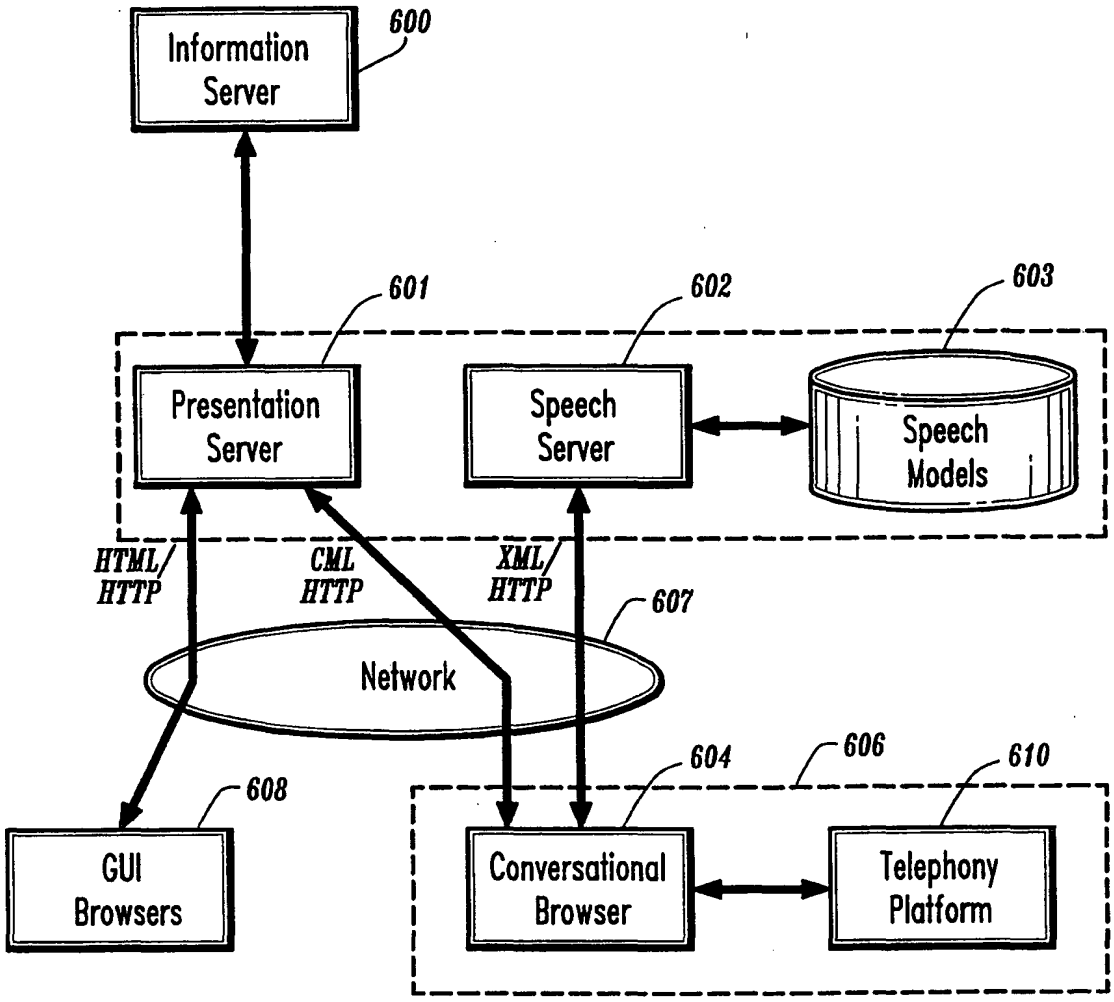

Information Server
600
601
602
603
Presentation Server
Speech Server
Speech Models
HTML/ HTTP
CML HTTP
XML/ HTTP
607
Network
604
606
610
608
GUI Browsers
Conversational Browser
Telephony Platform

FIG. 9

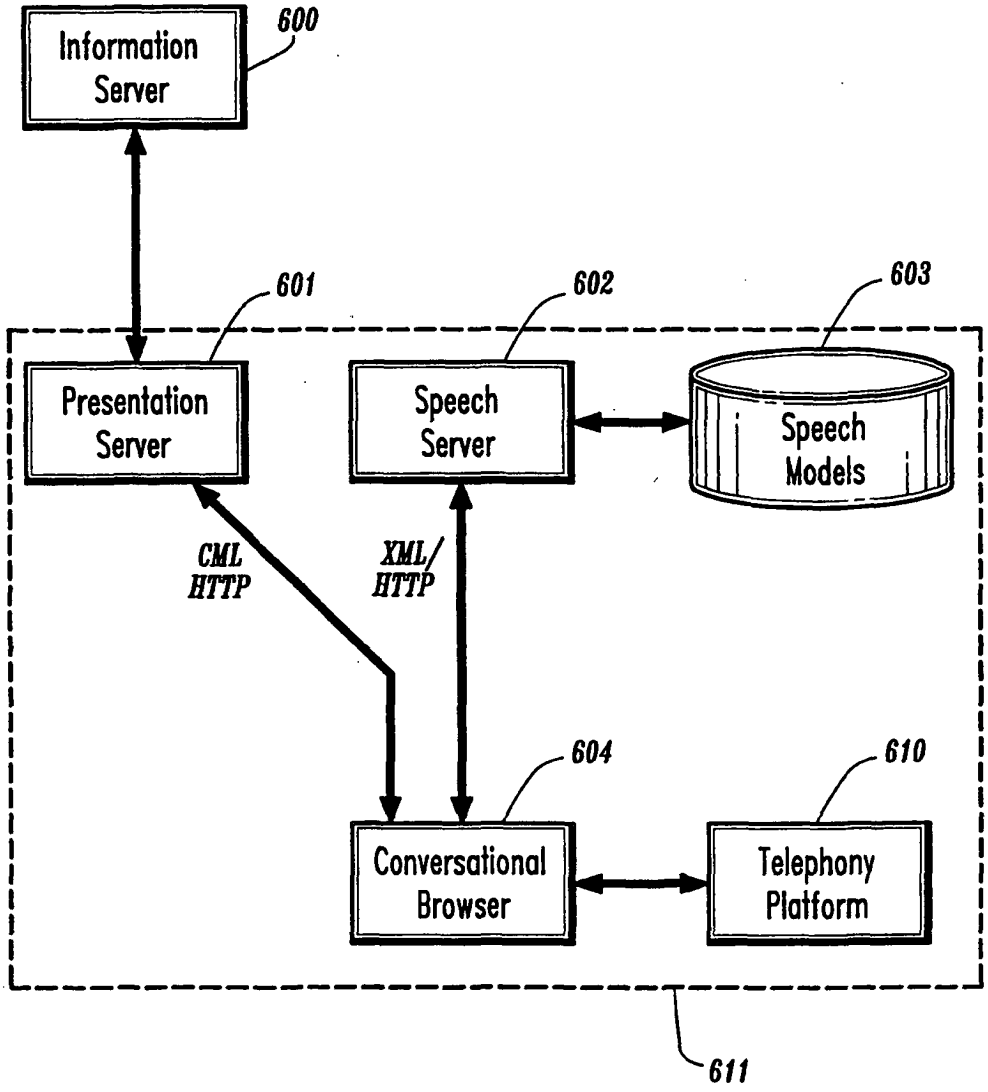

Information Server
600
601
602
603
Presentation Server
Speech Server
Speech Models
CML HTTP
XML/ HTTP
604
610
Conversational Browser
Telephony Platform
611

FIG. 10